Studies in Computational Intelligence 782

Saad Subair
Christopher Thron *Editors*

Implementations and Applications of Machine Learning

Springer

Saad Subair • Christopher Thron
Editors

Implementations and Applications of Machine Learning

Springer

Editors
Saad Subair
College of Computer Studies,
International University of Africa (IUA)
Khartoum, Sudan

Christopher Thron
Department of Science and Mathematics
Texas A&M University-Central Texas
Killeen, TX, USA

ISSN 1860-949X ISSN 1860-9503 (electronic)
Studies in Computational Intelligence
ISBN 978-3-030-37829-5 ISBN 978-3-030-37830-1 (eBook)
https://doi.org/10.1007/978-3-030-37830-1

Preface

Since people first started to construct computational machines that magically accept input, process it, and produce output, they have been inventing ways (a.k.a. "algorithms") to make these machines do the challenging jobs that they wished and dreamed of. Mathematicians who considered computational processes from a theoretical point of view invented and developed the concept of Turing machines, an abstract mathematical model of computation that helped scientists to understand and characterize algorithms that solve various problems. Researchers continued to push back the boundaries of what machines were capable of and envisioned machines that could actually think. The field of artificial intelligence (AI) has encompassed these efforts to make the machine capable of acting natural intelligence of human beings.

Within AI, machine learning is a prominent field. Machine learning refers to algorithms that construct mathematical models that can be used to make data-based decisions on optimization, clustering, classification, or prediction, where the decision process itself depends on the data. So machine learning resembles (and indeed overlaps with) statistics, which deals with the problem of finding predictive and/or fitting functions based on a sample data. Regression is one example of a topic within classical statistics that is also sometimes grouped under machine learning. However, machine learning goes farther than statistics by making use of *supervised learning,* in which pre-classified training data can be used to tune the algorithm for analyzing subsequent data [1].

Machine learning has become a fundamental tool in applied science. Many if not most state-of-the-art technological innovations now involve machine learning to a greater or lesser extent. Fields of its application are extremely diverse. In this book, we see the following applications: wireless networks, power systems design, image recognition, computer vision, biological learning, bioinformatics, data mining, and data verification.

This book provides clear, step-by-step explanations of successful practical applications of machine learning, at a level such that beginning practitioners in the field of machine learning (or advanced undergraduates in science and technology-related fields) can use these as prototypes for their own applications. Instructors may use these as examples in courses on machine learning.

This volume has several characteristics that set it apart from other books and articles in this area. First, the chapters have a dual focus: they give elementary explanations of established concepts and techniques, while also giving detailed presentations of contemporary applied research. Consequently the book provides a resource for machine learning students and other newcomers to the area, as well as for experienced practitioners who are interested in potential applications. Second, for many chapters documented code is available on the book's GitHub page:

https://github.com/chuks-ojiugwo/Implementations-and-Applications-of-Machine-Learning

Finally, many of the contributions represent work done in countries not ordinarily associated with high-tech research: Cameroon, Nigeria, and Sudan, along with South Africa, India, and the USA. This reflects how machine learning has leveled the playing field and enabled worldwide participation in the unfolding development of this exciting new area.

Machine learning is a vast area and includes a number of powerful techniques. Some of the techniques that are utilized in this volume are briefly presented as follows:

Genetic algorithms and swarm intelligence are biologically inspired approaches to optimization. Genetic algorithms loosely mimic the evolution of species via mutation and natural selection, while swarm intelligence involves a collection of "agents" that search out possible solutions and communicate their findings to each other. Chapters "Parallel 3-Parent Genetic Algorithm with Application to Routing in Wireless Mesh Networks" and "Application of Evolutionary Algorithms to Power System Stabilizer Design" in the current volume fall under this category, with applications to optimization in communications networks and power systems, respectively. Genetic algorithms, differential evolution, and population-based incremental learning (which is also discussed in chapter "Application of Evolutionary Algorithms to Power System Stabilizer Design") all fall under the category of evolutionary algorithms, in which populations of candidate solutions undergo successive modifications leading to better candidate solutions. These modifications can be considered as learned adaptations to the "environment" of possible solutions.

Gaussian mixture modeling is a clustering algorithm that produces a continuous probability distribution. Given a set of vector data, Gaussian mixture modeling approximates the empirical probability distribution with a sum of Gaussian distributions with specified means and covariances. Each Gaussian corresponds to a different cluster. When applied to time-dependent data, the means and variances of the Gaussian distributions are updated as the data changes. This technique is used for motion detection in the automatic sign language recognition system described in chapters "Automatic Sign Language Manual Parameter Recognition (I): Survey" and "Automatic Sign Language Manual Parameter Recognition (II): Comprehensive System Design" and is explained in chapter "Computer Vision Algorithms for Image Segmentation, Motion Detection, and Classification".

Support vector machines were originally designed for binary classification tasks. Beginning with the idea of linear separation using hyperplanes, support vector machines employ a "kernel trick" to enable separation using nonlinear hypersurfaces. Various techniques can be used to enable support vector machines to accomplish multi-way classification. Support vector machines are explained in chapter "Computer Vision Algorithms for Image Segmentation, Motion Detection, and Classification", and they are used for classification of hand shapes in the automatic sign language recognition system described in chapters "Automatic Sign Language Manual Parameter Recognition (I): Survey" and "Automatic Sign Language Manual Parameter Recognition (II): Comprehensive System Design".

Neural networks are computational structures inspired by brain function that can be trained to recognize patterns in inputs and produce desired outputs. They are composed of computational units called "neurons" that are arranged in layers, such that outputs of neurons in each layer are combined to form the inputs for successive layers. "Deep learning" makes use of neural networks with many layers. Most often, the neural networks used in deep learning are *convolutional neural networks*, whose structure reflects strong local correlations and translation invariance that is a property of many types of inputs (including images). Convolutional neural networks are used in the facial recognition system discussed in chapters "Overview of Deep Learning in Facial Recognition" and "Improving Deep Unconstrained Facial Recognition by Data Augmentation", as well as the plant image classification system presented in chapter "Improved Plant Species Identification Using Convolutional Neural Networks with Transfer Learning and Test Time Augmentation". A fairly recent development in the field of neural networks is the invention of *spiking neural networks*, which employ pulsed signals between neurons such that the pulse timing conveys information between the neurons. The functioning of these networks is much closer to the way the brain actually works than conventional neural networks, and they have the additional advantage of using much less power. Spiking neural networks are overviewed in chapter "Simulation of Biological Learning with Spiking Neural Networks", and specific examples are given for clarity.

Data mining involves the identification of patterns in large data sets. Important techniques within data mining include association rules and clustering. Association rules are used for instance in genetics, such that partial knowledge of a genome can be used to estimate the chances of the occurrence of other genes. A number of algorithms for finding association rules exist: a new binary-based algorithm is introduced in chapter "An Efficient Algorithm for Mining Frequent Itemsets and Association Rules" which has superior performance.

Statistical techniques are frequently used in machine learning. Indeed, machine learning is closely allied with statistics, and there is not a clear boundary between the two fields [1]. Classification is one of the key problems addressed by machine learning, and statistical techniques can be used to assess the quality of classification schemes. One such statistical technique (receiver operating characteristic curves) is discussed and applied in chapter "Receiver Operating Characteristic Curves in Binary Classification of Protein Secondary Structure Data".

Reinforcement learning/dynamic programming is an optimization technique from operations research that searches through the set of all possible solutions and makes use of information gained during the search to guide subsequent search progress. This method can be used when the possible solutions can be structured into a trellis-like format. Dynamic programming is introduced with a simple example in chapter "Budget Reconciliation Through Dynamic Programming", and a practical application is also given.

For completeness, we list some prominent topics in machine learning that are not discussed in this volume:

- Bayes classifiers and decision trees
- Clustering algorithms (including *k*-means and variants)
- Markov chain Monte Carlo
- Statistical learning

Brief summaries of the chapters are given as follows:

"Parallel 3-Parent Genetic Algorithm with Application to Routing in Wireless Mesh Networks" by Singh, Kumar, Singh, and Walia, proposes a new multi-population global optimization algorithm: the parallel 3-parent genetic algorithm (P3PGA). The performance of the new algorithm was compared with 16 other algorithms based on 30 benchmarks. P3PGA was found as the best-performing algorithm on 14 out of the 30 benchmark functions.

"Application of Differential Evolution to Power System Stabilizer Design" by Mulumba and Folly, focuses on optimization of power systems stabilizers (PSS). The chapter examines when there are power exchanges between large areas of interconnected power systems or when power is transferred over long distances under medium to heavy conditions. Conventional Power Systems Stabilizers (CPSS) performance deteriorates as the operating conditions change and hence requires re-tuning. This chapter describes the application of two evolutionary algorithms (differential evolution and population-based incremental learning) to optimize PSS parameters to provide adequate performance for a wide range of operating conditions.

"Automatic Sign Language Manual Parameter Recognition (I): Survey" by Ghaziasgar, Bagula, Thron, and Ajayi, surveys past work on the automatic recognition and transcription of sign language semantic information from monocular video. This complicated task involves a number of specific subtasks: The prior art presented in this chapter is improved upon in Chapter "Automatic Sign Language Manual Parameter Recognition (II): Comprehensive System Design" by Ghaziasgar, Bagula, and Thron, which describes key components of the design of a new video-based system for translation from sign language to English. Several of the key machine learning algorithms used in this new system are described in Chapter "Computer Vision Algorithms for Image Segmentation, Motion Detection, and Classification" by Ghaziasgar, Bagula, and Thron. Algorithms discussed include adaptive Gaussian thresholding and image thresholding for edge detection; the Viola Jones model for face detection; Gaussian mixture modeling for motion detection;

and histogram of oriented gradients descriptor and support vector machines for image classification.

"Overview of Deep Learning in Facial Recognition" by Ngezha, Fendji, and Thron, presents an overview of currently available deep neural network models in facial recognition. The chapter outlines the architectures and methods used by the best current models, and discusses performance issues related to the loss function, the optimization method, and the choice of training dataset. The discussion of facial recognition is continued in Chapter "Improving Deep Unconstrained Facial Recognition by Data Augmentation," which applies a powerful and versatile technique (data augmentation) to improve facial recognition for images obtained under uncontrolled lighting conditions.

"Plant Species Identification with Transfer Learning and Test Time Augmentation" by Igbineweka, Sawyerr, and Fasina, addresses the difficult task of identifying plant species, which is hard enough for botanists and virtually impossible for non-botanists. Many traditional machine learning techniques for plant identification rely on handcrafted features such as the shape, area, and perimeter of the plant, which tend to be error-prone and tedious to program. This chapter proposes a deep learning method for recognizing plant species. Three deep convolutional neural net architectures are trained using the concept of transfer learning, and their predictions averaged using ensemble learning. Further improvement in classification accuracy was obtained by applying test time augmentation.

"Simulation of Biological Learning with Spiking Neural Networks" by Ojiugwo, Abdallah, and Thron, gives an overview of spiking neural networks (SNN). SNNs are versions of artificial neural networks that are more biologically realistic and much less power-consuming than commonly used static models. As in actual brains, neurons signal each other by means of spikes (rather than constant inputs in conventional ANNs), and spike timing plays a key role in SNN functioning. This chapter describes the training of SNNs using the spike-dependent timing plasticity (STDP) algorithm and discusses an experiment that shows the ability of SNNs to learn to distinguish handwritten digits. An overview of current software and hardware simulators is also provided.

"Finding Association Rules for Large Datasets: An Efficient Binary-Based Approach" by Fageeri. Association rule mining (ARM) is an increasingly popular approach in data mining. This popularity is motivated by the fact that traditional statistical techniques, data management tools, and decision support systems are unable to handle enormous amounts of data. The key to effective application of association rules is finding a representation of database items that enable rapid identification and reduced memory. This chapter makes use of a binary representation of data, which makes it possible to employ very fast bitwise operations to speed up processing. This approach is verified on several benchmark datasets and shows that this binary-based approach outperforms other algorithms in terms of reduced execution time and memory usage.

"Receiver Operating Characteristic Curves (ROC) in Binary Classification of Protein Secondary Structure Data" by Subair and Thron. Protein secondary structure prediction is a fundamental step in determining the final structure and functions of a protein. Three states of secondary structures are identified, namely helices, strands, and coils, where coils typically comprise about 50% of the data. A binary classifier has been developed to group amino acids into two groups: coil and non-coil. This chapter applies receiver operating characteristic (ROC) analysis to analyze and interpret the results of the protein secondary structure classifier.

"Budget Reconciliation Through Dynamic Programming" by Laver, Brandt, and Thron, takes on a practical budgeting problem encountered by the US military. Because of complicated, interlocking financial systems, daily commits (orders) and obligations (account withdrawals) are not reconciled in detail. This chapter derives and implements an algorithm that takes a record of daily commits and obligations over a period of time and utilizes dynamic programming to identify the most likely matching between the two. The algorithm can also estimate the probability distribution of commit-to-obligation delays, thus making it a useful prediction tool. The algorithm can be adapted to a wide range of scenarios, and the performance has been verified via simulation as well as application to actual data.

Effective implementation of machine learning methods and algorithms has become an essential skill for students as well as researchers in both academia and industry. Great benefits may be reaped by learning from the research and advances of others within this field. This book provides an exceptional collection of the latest practical research and state-of-the-art algorithms associated with machine learning. We hope that this volume will inspire and equip readers to make further contributions to the machine learning revolution, which holds great promise for the betterment of our lives.

Killeen, TX Saad Subair
Killeen, TX Christopher Thron

Reference

1. D. Bzdok, N. Altman, M. Krzywinski, Statistics versus machine learning. Nat. Methods. (2018). https://doi.org/10.1038/nmeth.4642. Accessed 30 Oct 2019

Contents

Parallel 3-Parent Genetic Algorithm with Application to Routing in Wireless Mesh Networks

Amar Singh, Shakti Kumar, Ajay Singh, and Sukhbir S. Walia

1 Introduction

Genetic algorithms (GAs) are widely used computer-based search and optimization algorithms based on the mechanics of natural genetics and natural selection [1–5]. In the decade between 1950 and 1960 many researchers worked on evolutionary systems with the idea that evolution could be used as optimization approach for many engineering problems [4]. J. Holland introduced the concept of genetic algorithm in 1960 [5]. Usually genetic algorithms are based on two-parent genetic processes; however, some literature on multi-parent recombination can also be found in [6–9]. Mühlenbein and Voigt [6] presented the concept of gene pool recombination (GPR), and applied it to find solutions in a discrete domain. Eiben and Van Kemenade [7] proposed the concept of diagonal crossover as generalization of uniform crossover in GA and applied it to numerical optimization problems. Wu et al. [8] proposed multi-parent orthogonal recombination and applied it to find out the identity of an unknown image contour. The crossover operators used in those areas enabled significant improvements in search ability, although improvements were found to be highly problem dependent. Eiben et al. [9] proposed two multi-

A. Singh
Lovely Professional University, Phagwara, Punjab, India

S. Kumar (✉)
Panipat Institute of Engineering & Technology, Panipat, Haryana, India

A. Singh
Hochschule Wismar, University of Applied Sciences, Technology, Business and Design, Wismar, Germany

S. S. Walia
IK Gujral Punjab Technical University, Jalandhar, Punjab, India

© Springer Nature Switzerland AG 2020
S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_1

parent recombination mechanisms, namely gene scanning and diagonal crossover. They extensively tested their multi-parent algorithm on a variety of problems in numerical optimization, constrained optimization (traveling salesman problem) and constraint satisfaction (graph coloring). Compared to two-parent recombination, their algorithm achieved superior performance in optimizing the first four test functions of De Jong. For other problems the results were mixed; multi-parent crossover sometimes performed better and at times worse than classical two-parent recombination.

The parallel three-parent genetic algorithm presented in this chapter is based upon three-parent genetic processes in medical science and is very different from the approaches available in the literature. In medical science, a three-parent process has been used to prevent mitochondrial diseases in children of mothers with defective mitochondria [10–13]. In 2015, Dr. John Zhang and his team at the New Hope Fertility Center in New York City replaced the nucleus of a donor's egg cell with the nucleus of original mother, which was then fertilized with the father's sperm and implanted in the mother. The child that was subsequently born inherited mitochondrial DNA from the donor, besides nuclear DNA from the father and mother [13]. The concept of this three-parent genetic process is shown in Fig. 1. The P3PGA algorithm is in some respects a mathematical analogy of this practical process.

This chapter includes two different but related research investigations. The first is an evaluation of the performance of P3PGA on functions from an established test suite and comparison with other well-known recent algorithms. The second investigation concerns the application of P3PGA to an important and challenging practical problem, namely routing in wireless mesh networks (WMNs) [14].

This chapter is organized into six sections. Section 1 presents the motivation for this work; Section 2 discusses the working of P3PGA algorithm; Section 3 describes

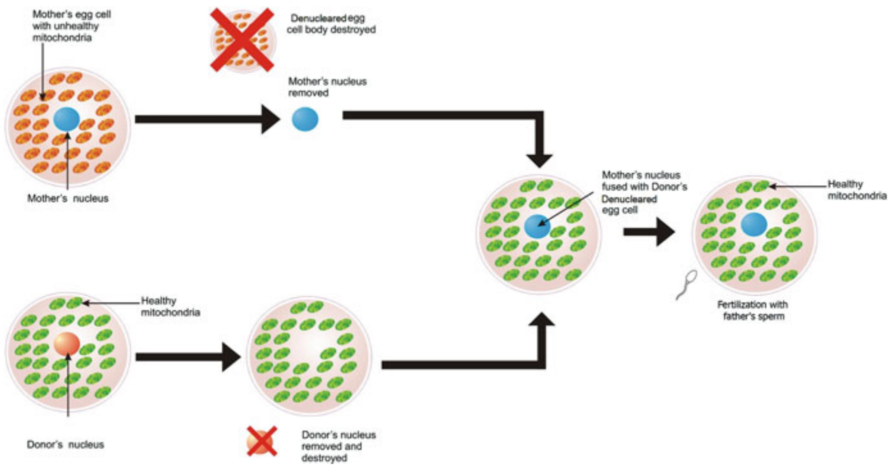

Mother's egg cell with unhealthy mitochondria
Denucleared egg cell body destroyed
Mother's nucleus removed
Mother's nucleus
Mother's nucleus fused with Donor's Denucleared egg cell
Healthy mitochondria
Fertilization with father's sperm
Healthy mitochondria
Donor's nucleus
Donor's nucleus removed and destroyed

Fig. 1 3-Parent Process adapted from Zhang et al. [13]

the simulation and performance of P3PGA on the 2014 Congress on Evolutionary Computation (CEC-2014) test bench suite compared to 16 other algorithms; Section 4 proposes a P3PGA-based minimal cost route evaluation approach for WMNs; Section 5 presents its implementation and performance on WMNs as well as a performance comparison with eight other algorithms found in the literature; and Sect. 6 summarizes conclusions.

2 P3PGA Algorithm

The Parallel 3-Parent Genetic Algorithm (P3PGA) is a multi-population algorithm in which evolution process takes place on several populations in parallel. It is based upon the single-population three-parent genetic algorithm (3PGA) [15, 16].

A pseudocode for the proposed P3PGA approach is as given in the listing below in Algorithm 1. In this algorithm we initially create several populations of equal size: these are called 2-parent populations. To generate a 3-parent population from a 2-parent population we applied the "mitochondrial change" to each individual by adding a small random number to each gene of each individual in the population. The fittest individual (best solution) within a population is called the *local elite* of that specific population. The best of all local elites is called the *global elite*. All genes for all individuals in all the populations are moved towards the corresponding gene of the global best candidate solution with low probability. This guides every candidate solution towards the global best solution.

For the better understanding of the algorithm, we describe its operation on a very simple example where we wish to search a 4-digit quantity such that the sum of 4 digits is maximal, given that each digit is an integer between 0 and 9 (the obvious answer is 9999). We proceed in such a way that the reader will be able to track progress towards this solution as the algorithm progresses. The algorithm parameters are chosen as shown in Table 1.

Step 1 Randomly create 3 two-parent populations, where each population consists of three candidate solutions and each candidate solution is an array of four digits (genes) having the form (d_1, d_2, d_3, d_4):

TwoParentPop(1): [(3,6,1,7), (5,2,6,3), (4,6,5,3)]
TwoParentPop(2): [(5,3,6,2), (5,6,7,8), (8,2,7,6)]
TwoParentPop(3): [(7,4,5,1), (5,6,1,3), (3,8,4,9)]

We may rewrite these three populations in matrix format, for example:

$$\text{TwoParentPop}(1) = \begin{bmatrix} 3 & 6 & 1 & 7 \\ 5 & 2 & 6 & 3 \\ 4 & 6 & 5 & 3 \end{bmatrix}$$

TwoParentPop(2) and TwoParentPop(3) may be expressed similarly as

Algorithm 1: P3PGA algorithm

```
begin
Generate NP populations each of size N candidates randomly, every candidate consisting of NG genes;
for gen = 1:Number of generations do
   for i = 1:NP do
      Effect Mitochondrial Change to $i^{th}$ 2-parent (2-P) population to Generate an $i^{th}$ 3-Parent (3-P) population
      Combine the 2-P and 3-P populations and select the N best individuals.
      Find and record the globally best solution.
      Generate a new 2-P population using general genetic process (using GA)
         (a) Select fit individuals for recombining/breeding.
         (b) With high probability recombine parents/perform cross-over.
         (c) With low probability mutate offspring.
         (d) Select the N best individuals from among parents and offspring...
      Check bounds violation and correct if needed.
end for (i)
Check the fitness of all the individuals of all the populations and select/update the globally best $g_{best}$ candidate and its fitness;
for i = 1: NP do
   for j = 1: N do
      for k = 1: NG do
         With a fixed small probability replace gene $k$ of individual $j$ in population $i$ with (individual(j,k) + $g_{best}$(j,k))/2;
      end for (k)
   end for (j)
end for (i)
end for (gen)
end
```

Table 1 Parameter values for illustrative example of P3PGA

Parameter description	Symbol	Value
Number of populations	NP	3
Population size	N	3
Number of genes in each individual	NG	4
Gene values	d_j $(j = 1 \ldots 4)$	0 thru 9
Crossover probability		0.9
Mutation probability		0.1

$$\text{TwoParentPop}(2) = \begin{bmatrix} 5 & 3 & 6 & 2 \\ 5 & 6 & 7 & 8 \\ 8 & 2 & 7 & 6 \end{bmatrix}$$

and

$$\text{TwoParentPop}(3) = \begin{bmatrix} 7 & 4 & 5 & 1 \\ 5 & 6 & 1 & 3 \\ 3 & 8 & 4 & 9 \end{bmatrix}$$

Step 2 For each gene of every individual in each of the populations, effect a mitochondrial change by adding a random number to each gene. In our case, each gene change is generated as a uniformly distributed random integer between −2 and 2. This may be implemented by generating a change matrix for each population, adding the change matrix to the population matrix, and truncating so that the gene values remain within the range from 0 to 9. For example, suppose the change matrix for the first population is given by:

$$\text{Change}(1) = \begin{bmatrix} 2 & 0 & -2 & 1 \\ 1 & -1 & 2 & 1 \\ 0 & 2 & 2 & 2 \end{bmatrix}$$

Let us further assume that the randomly generated mitochondrial changes for the second and third populations are as given below:

$$\text{Change}(2) = \begin{bmatrix} -2 & 0 & -1 & 1 \\ 1 & 2 & 2 & -1 \\ 0 & 2 & 2 & 2 \end{bmatrix}; \quad \text{Change}(3) = \begin{bmatrix} -2 & 0 & -1 & 2 \\ 1 & 2 & 2 & -1 \\ 0 & 1 & -2 & 2 \end{bmatrix}$$

Using 2-parent population and Change matrices for each population we compute the 3-parent populations as follows:

$$\text{3\_Parent\_Population}(n) = \text{2\_Parent\_Population}(n) + \text{Change}(n)$$

Whenever this formula produces an entry in 3_Parent_Population(n) that is less than 0 or greater than 9, it is replaced with 0 or 9, respectively.

Step 3 The three-parent populations are then combined with their corresponding two-parent populations to form populations that are twice as large. In our example, these combined populations are given by three 6×3 matrices (matrix rows are separated by semicolons)

Population 1: [3 6 1 7; 5 2 6 3; 4 6 5 3; 5 6 0 8; 6 1 8 4; 4 8 7 5]
Population 2: [5 3 6 2; 5 6 7 8; 8 2 7 6; 3 3 5 3; 6 8 9 7; 8 4 9 8]
Population 3: [7 4 5 1; 5 6 1 3; 3 8 4 9; 5 4 4 3; 6 8 3 2; 3 9 2 9]

We choose the three individuals for each population with highest fitness, which are then sorted in decreasing order of fitness. Table 2 shows the results of this operation.

Table 2 Populations, individuals, and their Fitness

Pop. No.	Individual No.	Individual	Fitness	Pop. No.	Individual No.	Individual	Fitness
1	1	4 8 7 5	24	3	1	3 8 4 9	24
	2	5 6 0 8	19		2	3 9 2 9	23
	3	6 1 8 4	19		3	6 8 3 2	19
2	1	6 8 9 7	30				
	2	8 4 9 8	29				
	3	5 6 7 8	26				

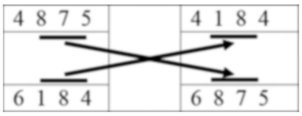


Fig. 2 Single-point crossover operation

Step 4 The conventional genetic algorithm operations of recombination (crossover) and mutation are now performed within the resulting populations.

Step 4a. Usually fitness of an individual is used as selection criterion for crossover. Various selection strategies may be used, including roulette wheel selection, stochastic universal sampling, truncation selection, tournament selection, and so on (interested reader may refer to [17–19] for details). We have used stochastic universal sampling (SUS) for selection of parents for recombination. In our example, we may suppose that individuals 1 and 3 in population 1 are selected, and a crossover operation is performed on the two individuals with high probability (usually between 0.75 and 0.9). A typical example of a crossover operation is shown in Fig. 2.

The operation shown is a single-point crossover with crossover point after first gene. Crossover can be single point, two point, or *n*-point. If crossover does not take place, then both individuals are passed on as offspring.

Step 4b. Following crossover, mutation is performed. In the case of real-valued genes, one way to perform mutation involves replacing the current gene with a randomly generated value from within the universe of discourse of that gene with a low probability. Both the probability of mutating a gene (mutation rate) and the distribution of changes for each mutated gene must be specified. For example, the change may be determined as a uniform random number in an interval $[-a, a]$. In our example, we have taken $a = 2$ and the mutated genes are rounded off to the nearest integer.

Step 4c. Following crossover and mutation, the offspring are grouped together with their parents for each population, and the three fittest from each population are chosen as the next generation. Note each generation of each population always has the same number of individuals (equal to *N*, which is one of the algorithm's basic parameters). Let us assume that the individuals 4 8 7 5 and 5 6 0 8 were selected for recombination assuming the crossover point was after first two genes. Thus, the combination produced two offspring, i.e., 4 8 0 8 and 5 6 7 5. Let us further assume that individual 6 1 8 4 was passed as it is as an offspring.

Table 3 Evolving new population after crossover and mutation

Current population	Offspring (after crossover and mutation)	New population (weak individuals replaced with stronger offspring)
4 8 7 5 (24)	4 8 0 7 (19)	5 8 7 5 (25)
5 6 0 8 (19)	5 8 7 5 (25)	4 8 7 5 (24)
6 1 8 4 (19)	6 0 8 4 (18)	5 6 0 8 (19)

Table 4 Local best (elites) and global best

Population number	Local best (ℓ_{best})/Elite	Fitness
1	5 8 7 5	25
2	6 8 9 7	30
3	5 9 2 9	25

Let us further assume that the mutation operator mutated first offspring to 4 8 0 7, second offspring to 5 8 7 5, and the third offspring to 6 0 8 4. Replacing the weaker parents with the stronger offspring produces the results as shown in Table 3.

Step 4d. Compute elites (local best) and global best:

Global best (g_{best}) candidate solution after one generation is: "6 8 9 7" with fitness values of 30 ($6 + 8 + 9 + 7 = 30$). The computation of elite (local best) is for better understanding of the algorithm only. For better code efficiency we can directly compute global best from the evolved generations. However, for algorithm 2, which is based upon algorithm 1, computation of elites (local best of each population) is essential (Table 4).

Step 5 After a predetermined number of generations, with a given probability we replace *i*th gene of every individual with a gene whose value is the average value of the *i*th gene of individual and *i*th gene of g_{best}, i.e.,

$$\text{individual}(i) \leftarrow (\text{individual }(i) + g_{\text{best}}(i))/2$$

Table 5 summarizes the results of first iteration of computer implementation of example using the P3PGA algorithm.

Continuing further execution of the program we find that our algorithm reached the best result in about 15 iterations. Figure 3 shows number of generations (iterations) versus fitness for our example.

3 Simulated Performance, Results, and Discussion

We implemented the proposed P3PGA algorithm in MATLAB on a Core i7 @ 2.2GHz based laptop with 8GB RAM and tested its performance on 30 functions from the CEC-2014 test bench. To evaluate the performance of P3PGA we used

Table 5 First iteration and corresponding output of each major step of computer implementation of P3PGA for the example under consideration

Population no.	#1				#2				#3			
Initial	6	6	4	4	1	5	9	1	7	4	2	2
2_Parent (2P)	5	2	8	2	2	9	8	4	7	5	9	5
Population	8	4	7	5	6	6	7	1	9	6	1	6
Mitochondrial	−1	−2	−1	0	1	1	0	0	−1	−1	−1	1
Change	2	1	−1	−1	0	−1	0	1	−2	1	1	0
	0	−1	−2	−2	−1	0	1	0	−1	0	1	−1
3-parent (3P)	5	4	3	4	2	6	9	1	6	3	1	3
Population	7	3	7	1	2	8	8	5	5	6	9	5
	8	3	5	3	5	6	8	1	8	6	2	5
Best *N* individuals	8	4	7	5	2	9	8	4	7	5	9	5
Selected from 2P & 3P	6	6	4	4	2	8	8	5	5	6	9	5
In descending order	8	3	5	3	6	6	7	1	9	6	1	6
Selection of parents	8	3	5	3	2	9	8	4	5	6	9	5
For recombination	6	6	4	4	6	6	7	1	9	6	1	6
	8	3	5	3	6	6	7	1	9	6	1	6
Population	8	3	4	4	2	9	8	1	5	6	9	6
After crossover	6	6	5	3	6	6	7	4	9	6	1	5
	8	3	5	3	6	6	7	1	9	6	1	6
Population	8.0	3.0	5.0	4.0	2.0	9.0	8.0	1.0	5.0	6.0	9.0	5.0
After mutation	6.0	5.8	4.1	3.0	6.1	6.0	7.0	3.9	9.0	6.0	2.1	6.0
(genes to be rounded off)	8.0	3.0	5.0	3.0	6.0	6.0	7.0	1.0	7.8	6.0	1.0	6.0
Population after replacing weak	8	4	7	5	2	9	8	4	7	5	9	5
parents with strong	8	3	5	4	6	6	7	4	5	6	9	5
Offspring	6	6	4	4	2	8	8	5	9	6	2	6

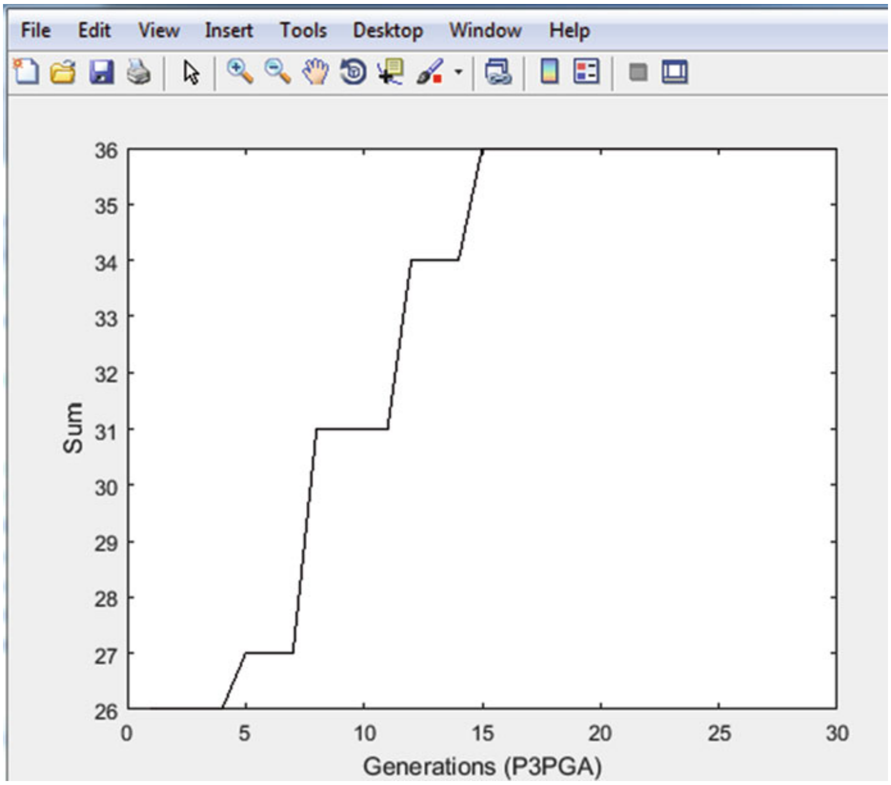


Fig. 3 Generation number versus fitness of globally best result for simple P3PGA example

$N = 10$ populations and NC = 20 candidate solutions. Our P3PGA results were compared to results obtained from CEC-2014 for 16 other algorithms: the United Multi-Operator Evolutionary Algorithms (UMOEAS) [20], LSHADE [21], Differential Evolution with Replacement Strategy (RSDE) [22], Memetic Differential Evolution Based on Fitness Euclidean-Distance Ratio (FERDE) [23], Partial Opposition-Based Adaptive Differential Evolution Algorithms (POBL_ADE) [24], Differential Evolution strategy based on the Constraint of Fitness values classification (FCDE) [25], Mean-Variance Mapping Optimization (MVMO) [26], RMA-LSCh-CMA [27], Bee-Inspired Algorithm for Optimization (OptBees) [28], Simultaneous Optimistic Optimization (SOO) [29], SOO+ Bound Optimization BY Quadratic Approximation (SOO + BOBYQA) [29], Fireworks Algorithm with Differential Mutation (FWA-DM) [30], algorithm Based on Covariance Matrix Leaning and Searching Preference (CMLSP) [31], Gaussian Adaptation Based Parameter Adaptation for Differential Evolution (GaAPADE) [32], Non-Uniform Mapping in Real-Coded Genetic Algorithms (NRGA) [33], and DE_b6e6rlwithrestart [34]. The

MATLAB code for the compared algorithms and the link to algorithm performance results may be obtained from http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2014/CEC2014.htm. For P3PGA we conducted 20 trials for each of the 30 functions, and took the mean error of all 20 trials as the performance measure.

The performance of P3PGA along with 16 other algorithms is given in Table 6 (see Appendix 1). Results of the comparison between algorithms are summarized in Table 7. P3PGA gave the unmatched best performance for 12 functions (f5, f9, f10, f11, f14, f15, f16, f19, f21, f24, f25, and f27) (Table 8). For f3 the performance of P3PGA was equaled by UMOEAS, RSDE, FCDE, DE_ b6e6rlwithrestart, LSHADE, MVMO, FWA-DM, and GaAPADE algorithm; and for f8 P3PGA's performance was matched by UMOEAS, FERDE, DE_b6e6rlwithrestart, GaAPADE, LSHADE, MVMO, OptBees, and RMA-LSCh-CMA. Altogether, P3PGA was the top-ranked algorithm, while UMOEAS was second.

4 P3PGA for Minimal Cost Route Evaluation

A wireless mesh network (WMN) can be mathematically represented as a set of "nodes" or points in the two-dimensional plane. These nodes represent the positions of clients, routers, and gateways that receive and retransmit communications signals. Naturally, the devices represented by nodes all have limited communication range.

Table 7 Comparative performance of P3PGA on CEC-2014 benchmarks

Algorithm	Best (unmatched)	Best (matched)	Total best	Rank
P3PGA	12	2	14	1
UMOEAS	1	8	9	2
LSHADE	2	4	6	3
DE_b6e6rlwithrestart	1	5	6	3
GaAPADE	1	4	5	4
SOO + BOBYQA	0	5	5	4
MVMO	1	3	4	5
RMA-LSCh-CMA	0	4	4	5
SOO	0	4	4	5
FCDE	0	3	3	6
RSDE	0	3	3	6
CMLSP	0	3	3	6
FERDE	0	2	2	7
POBL_ADE	1	0	1	8
FWA-DM	0	1	1	8
OptBees	0	1	1	8
NRGA	0	0	0	–

Table 8 Number of functions for which P3PGA gave the best performance

Functions for which P3PGA was a clear winner	f5, f9, f10, f11, f14, f15, f16, f19, f21, f24, f25, f27
Function for which P3PGA was a joint winner	f3, f8

In order to send a signal from a source node to a destination node, a *route* (or *path*) consisting of intermediate nodes must be found such that the signal from the source node can be successively received and retransmitted until it reaches the destination node. The process of determining the end-to-end route between a source node and a destination node is referred to as "routing." An optimal route will be one that minimizes cost, where cost is defined in terms of a routing metric. There are many possible routing metrics that appear in the literature, including minimum hop count, per hop Round Trip Time (RTT) [35], Per-Hop Packet Pair Delay (PktPair) [36], Expected Transmission Count (ETX) [37], Expected Transmission Time (ETT), Weighted Cumulative ETT (WCETT) [38], Expected Transmission on a Path (ETOP) [39], Effective Number of Transmission (ENT) and Modified Expected Number of Transmissions (mETX) [40], Metric of Interference and Channel Switching (MIC) [41], Bottleneck Link Capacity (BLC) path metric [42], cross layer link quality and congestion aware (LQCA) metric [43], and interference aware low overhead routing metric [44]. In this chapter, we use an integrated link cost function to evaluate route cost: for details see [45].

The adapted P3PGA algorithm that was used for finding optimal routes for WMNs is outlined in Algorithm 2 below. The algorithm follows all the steps of the general P3PGA algorithm described in Algorithm 1 in the previous section. These steps are described in more detail in the following paragraphs.

The first step in the algorithm is to determine initial populations of possible routes. This is done by means of an *adjacency matrix*, which is a (number of nodes) by (number of nodes) square matrix of 0's and 1's. The (*i,j*)th entry of the matrix is 1 if nodes *i* and *j* have a possible connection, and 0 otherwise. Using the adjacency matrix a set of route populations is generated, where each population has the same number of routes. These are the initial 2-parent populations. Next, we generate a 3-parent population from each of the current 2-parent populations and combine these two populations as in Algorithm 1. The new population is evolved from the old population using the crossover approach as given in [46]. Each 3-parent population is obtained by applying the following rules to all routes in the 2-parent population:

(a) Beginning with second node, check the location of the NDth node of the local elite in the current path, where ND is defined so that nodes 2 . . . ND-1 of the local elite are not in the current path, but ND is in the current path.
(b) If the NDth node lies in first half of the current path, then follow the steps as given below:

I. Retain nodes of current individual up to ND and call it partial route1.
II. From the elite extract all nodes from the node after “ND” to the terminal node and call it partial route2.
III. Combine partial route1 and partial route2 to form a new path.

(c) If the “NDth” node of the local elite lies in the second half of the current path, then follow the steps as given below:

I. Retain the local elite up to “ND” and call it partial route1.
II. From the current path extract all nodes from the node after “ND” to the terminal node and call it partial route2.
III. Combine partial route1 and partial route2 to form a new route.

For example, in a WMN, let node number 1 represent the source node and node number 10 represent the terminal node. Suppose that the following two-parent population of routes exists between source and terminal node:

2-parent population 1 : [1 4 5 8 9 10; 1 3 2 7 12 8 11 9 10; 1 6 7 4 13 12 5 9 10]

In this population the local elite (shortest path) is: 1 4 5 8 9 10

Since the local elite is the fittest route (minimal cost path as per hop count method) we would not apply mitochondrial change to this elite route.

In the second route (2-parent route 2) we find that node 8 is the first node in the route that is shared with the local elite. This node lies in the second half of 2-parent route 2. Hence, we would retain the local elite up to node 8 and denote it as partial route 1:

Partial route 1 : [1 4 5 8]

From the current route (2-parent route 2), we extract all nodes starting from the node after 8 up to the terminal node, and denote it as partial route 2:

Partial route 2 : [11 9 10]

After combining partial route 1 and partial route 2 we get a new 3-parent route:

3-parent route 1 : [1 4 5 8 11 9 10]

Similarly, we make the mitochondrial change in 2-parent route 3 as follows. We first identify that the first node in the route that is shared with the elite is node 4, which lies in the first half of the current route. So we retain current route up to node 4 and denote it as partial route 1:

Partial route 1 : [1 6 7 4]

From the elite we extract all nodes after node 4 up to the terminal node, and denote it as partial route 2.

$$\text{Partial route 2} : [5\ 8\ 9\ 10]$$

After combining partial route 1 and partial route 2 we obtain a second 3-parent route as follows:

$$\text{3-parent route 2} : [1\ 6\ 7\ 4\ 5\ 8\ 9\ 10]$$

We combine the 2-parent routes with the newly evolved 3-parent routes into a single combined population:

$$[1\ 4\ 5\ 8\ 9\ 10;\quad 1\ 3\ 2\ 7\ 12\ 8\ 11\ 9\ 10;\quad 1\ 6\ 7\ 4\ 13\ 12\ 5\ 9\ 10; 1\ 4\ 5\ 11\ 9\ 10;\\ 1\ 6\ 7\ 4\ 5\ 8\ 9\ 10]$$

The algorithm then evaluates the fitness of all routes in this combined population, dropping the weaker individuals and retaining the fittest N individuals, thus maintaining a constant population size.

Once this is completed, the standard genetic processes of crossover and mutation can be performed just as demonstrated in Algorithm 1, to obtain optimal solutions for all populations. The optimum of these local optima gives the current global optimum, which may then be used to update the routing table entry for the given source and destination node, so that subsequent data transfer between these two nodes may take place on the minimal cost routes. Being parallel in nature the convergence time of this algorithm is expected to be quite small.

5 Implementation and Performance of the Proposed Approach

To evaluate the comparative performance of the proposed P3PGA-based minimal cost route evaluation approach for WMNs, we implemented all the approaches in MATLAB and simulated for 100, 500, 1000, 2000, and 2500 node client WMNs. Parameters for the different WMNs are shown in Table 9. For each WMN, node locations were randomly generated within the specified area. To evaluate the performance of all approaches on 100, 500, 1000, and 2000 node client WMNs we conducted 10 trial sets, one set for a given timing constraint. On 2500 node client WMNs we considered 17 trial sets. For all networks, each trial set consists of 20 trials: to test algorithm performance in a dynamic environment, the node locations were randomly regenerated for each trial. In total, 1340 trials were conducted.

Algorithm 2: P3PGA approach for dynamic optimal cost route evaluation

```
Begin
/* Adjacency matrix: matrix of neighbor nodes of each node */
/* Variables:
    pop_mat: Populations of routes;
    path_mat: Matix whose rows are the routes in a given population;
    NP: Number of populations,
    N: Number of routes per population
    Path_nodes: Number of nodes in a route.
*/
Calculate N routes using adjacency matrix.
Evaluate fitness of all routes in all populations.
Determine the local best routes $\ell_{best}(i)$, $i = 1 \ldots NP$ for all populations i.
Record the global best ($g_{best}$) route from amongst all the local best routes;
for Gen = 1: Number of Generations do
   for pop = 1: NP do
      /* 3 Parent Population generation starts */
      3P_path = pop_mat(pop).path_mat
      local_elite = $\ell_{best}$(pop)
for i = 1:N do
   SE = number of nodes in local_elite
         if 3P_path(i) ! = local_elite then
            for j = 2:SE-1 do
               LOE = location of local_elite(j) in 3P_Path(i).
               if LOE > 0
                  if LOE > path_mid then /* path_mid = half of i$^{th}$ path*/
                     partial_route1 = first j nodes of local_elite
                     partial_route2 = nodes from LOE+1 to target node in 3P_path(i)
                  else
                     partial_route1 = first LOE nodes of 3P_path(i)
                     partial_route2 = nodes from j + 1 to target node in local_elite
                  end if
                  new_3P_path = concatenation of partial_route1 and partial_route2
                  Append new_3P_path to 3P_path
                  Break
               end if
            end for (j)
         end if
      end for (i) /* 3 Parent Population generation Ends */
      /* Generation of 2-parent population from 3-parent population starts */
      Evaluate fitness of all paths in 3P_path, sort from best to worst and select the N best paths
      Select the fit paths for recombining/breeding;
      With high probability recombine parents/perform cross-over.
      With low probability mutate paths.
      Evaluate fitness and select local_best.
      Replace weak paths by stronger paths keeping the path_mat size fixed at N;
      pop_mat(pop).path_mat = 3P_Path;
   end for (pop) /*Generation of 2 Parent Population from 3 Parent Population Ends */
   From amongst the NP local best candidates select the global best candidate $g_{best}$;
```

```
    /*move all routes of all populations towards global best*/
    for i = 1:NP do
    for j = 1:N do
            for k = 1:Path_nodes(j) do
                With a given probability replace k^th node of j^th route with a node of $g_{best}$ using combination operation.
    end for (k)
            end for (j)
    end for (i)
end for (gen)
```

Table 9 Architectural details of client WMNs used in simulations

No. of nodes	Area (m^2)	Radio range	Timing constraint (in seconds)
100	500 × 500	150	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
500	500 × 500	150	0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0
1000	1000 × 1000	250	0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0
2000	2000 × 2000	250	1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5
2500	2000 × 2000	250	2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0

5.1 Comparative Performance of 100 Node Client WMNs

For 100 node client WMNs we evaluated the performance of nine approaches. The unmatched performances of all the nine approaches are shown in Table 10 and Fig. 4. Table 10 shows the total performance (matched and unmatched) of nine approaches. From the results, we observe that ACO and DSR are unreliable protocols for the given network scenarios because most of the time these protocols failed to discover any feasible routes between source-terminal pair. Being a proactive approach, the BAT approach successfully discovered feasible routes but failed to produce an unmatched optimal cost route in any of the trials.

Table 10 shows that for the timing constraint of 0.1 second, AODV produced a minimum cost route 7 + 5 = 12 times, where the first operand (7) indicates that 7 times AODV generated an unmatched optimal cost route, and the second operand (5) indicates that 5 times other algorithms obtained the same minimum cost (in this case, the other algorithms are GA, BBBC, FA, and P3PGA). The second and third place algorithms were BBBC (5 + 5 = 10 times) and P3PGA (3 + 5 = 8 times).

For the timing limit of 0.2 second, P3PGA produced minimum cost route 8 + 2 = 10 times, AODV 6 + 2 = 8 times, BBBC 3 + 2 = 5 times, and BBO produced minimum cost route 1 + 2 = 3 times. Figure 4 shows that for 100 node networks and for timing constraints less than 0.5 s (except 0.2 s) AODV performs better than all other algorithms. For timing constraint of 0.5 and 0.6 s the performance of GA is best. For timing constraint of 0.7 s BBBC and P3PGA

Table 10 Comparative performance of P3PGA on 100 node client WMNs

	Timing constraints									
Algorithm	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
AODV	7 + 5	6 + 2	5 + 9	5 + 5	1 + 9	2 + 10	2 + 11	2 + 10	4 + 0	3 + 9
DSR	16 FAIL	17 FAIL	7 FAIL	15 FAIL	1 + 4 FAIL	10 FAIL	FAIL	4 FAIL +2	12 FAIL	9 FAIL
ACO	19 FAIL	12 FAIL	16 FAIL	13 FAIL	8 FAIL	2 FAIL	5 FAIL	3 FAIL	5 FAIL	11 FAIL
GA	0 + 5	0 + 2	0 + 9	2 + 5	3 + 12	4 + 10	1 + 11	1 + 10	1 + 1	2 + 9
BBO	0 + 1	1 + 2	0 + 2	0 + 5	1 + 0	0 + 10	1 + 0	0 + 10	1 + 0	0 + 9
BBBC	5 + 5	3 + 2	2 + 9	4 + 5	1 + 9	1 + 10	3 + 11	1 + 10	3 + 1	1 + 9
FA	0 + 5	0 + 1	0 + 9	1 + 1	1 + 9	1 + 10	0 + 9	0 + 10	2 + 1	0 + 9
P3PGA	3 + 5	8 + 2	4 + 9	3 + 5	2 + 12	2 + 10	3 + 11	4 + 10	7 + 1	5 + 9
BAT	0 + 1	0 + 0	0 + 2	0 + 0	1 + 0	0 + 10	0 + 5	0 + 10	0 + 3	0 + 6

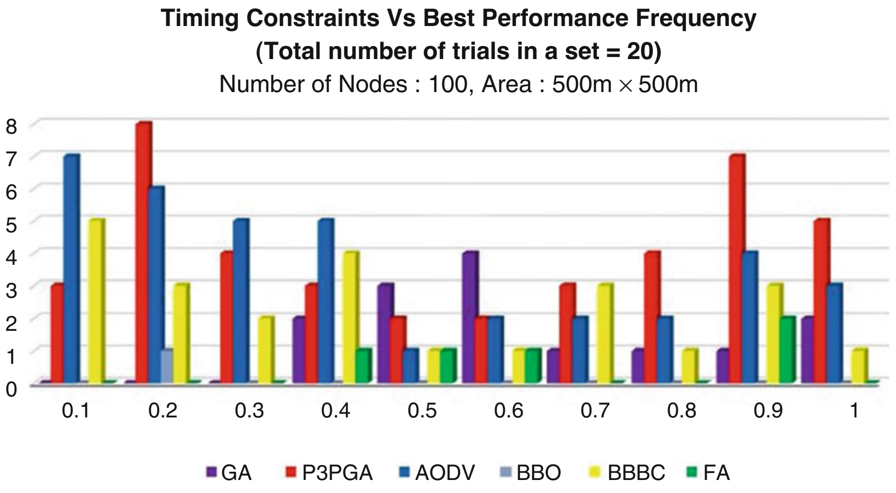


Fig. 4 Comparative unmatched best performance of 100 node client WMNs

both give best performance. As the timing constraint is further increased, more computing time is allocated to the P3PGA algorithm, so that results improve with increasing timing constraint. In terms of producing optimal cost routes, for timing constraints of 0.2, 0.8, 0.9, and 1.0 s, P3PGA algorithm outscores all other algorithms.

5.2 *Comparative Performance of 500 Node Client WMNs*

For 500 node client WMNs we evaluated the performance of all given 9 optimal route evaluation approaches. The performance results of the all approaches are given in Table 11 and Fig. 5. Figure 5 shows the unmatched performance of all nine approaches and Table 11 shows the total performance of all considered approaches. From results we observe that on the given WMN scenario up to 3.5 s timing constraints the AODV routing protocol outperforms all its competitors. But after

Table 11 Comparative performance of P3PGA on 500 node client WMNs

	Timing constraints									
Algorithm	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
AODV	16	10+ 1FAIL+ 5	15	14	9	19	11	6	10	4
DSR	–	–	–	–	–	–	–	–	–	–
ACO	–	–	–	–	–	–	–	–	–	–
GA	0	0 + 5	1 + 1	0	4 + 1	0	0	1 + 2	0	2
BBO	0	0 + 5	0	0	0	0	0	1 + 2	0	1
BBBC	0	1 + 5	1	2	4	1	2	1 + 2	0	3
FA	0	1 + 5	0	1	0	0	1	0 + 2	0	0
BAT	0	0 + 5	0	0	0	0	0	0 + 2	0	0
P3PGA	4	3 + 5	2 + 1	3	2 + 1	0	6	9 + 2	10	10

"–" means failed to produce route in any of the trials

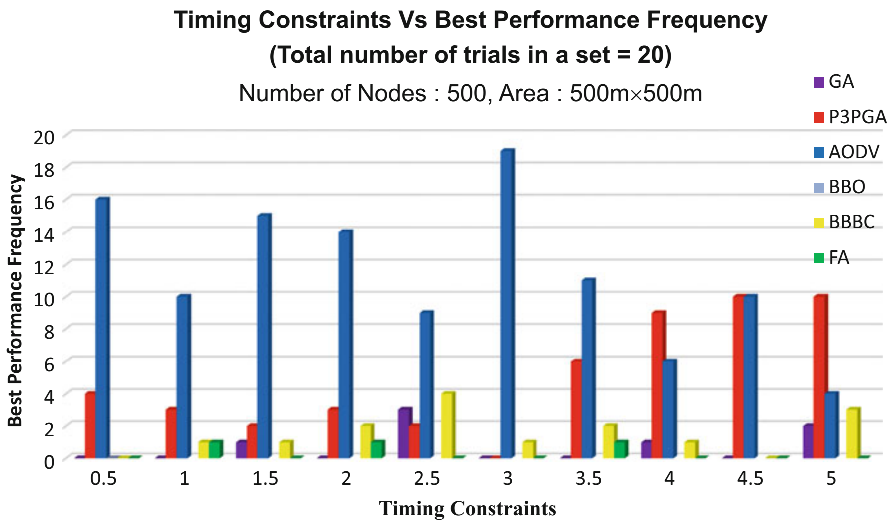


Fig. 5 Comparative unmatched best performance of 500 node client WMNs for different timing constraints

3.5 s all other approaches also started to perform. On the timing constraint of 4.0 s P3PGA produced minimum cost route 9 + 2 = 11 times, AODV 6 times, BBBC 1 + 2 = 3 times, Firefly 1 + 2 = 3 times, and GA produced minimum cost route 1 + 2 = 3 times only. With 5 s timing limit P3PGA generated minimum cost route 10 times, AODV 4 times, GA 2 times, BBO 1 time, and BBBC produced minimum cost route 3 times.

5.3 *Comparative Performance of 1000 Node Client WMNs*

Table 12 and Fig. 6 present the simulated performance for 1000 node client WMNs. From the performance we observe that DSR and ACO approaches failed to discover the route for the given timing constraint in any of the trial sets. Up to 2 s timing limits AODV also failed to discover any of the routes. As shown in Fig. 6, P3PGA outperforms other 8 approaches for the timing constraints of 0.5, 1.0, 1.5, 2.5, and 3.0 s. With timing constraint of 2.0 s P3PGA and BBBC gave the same best performance. Further, we also observed that after the 3.0 s timing constraint the performance of AODV improved considerably to the extent that it outperformed all other 8 approaches. Hence, for the given WMN scenarios AODV is unsuitable approach if the network size is 1000 node with allowable computing time less than 2 s. If timing constraint could be relaxed beyond 3 s, then the AODV gives the best performance.

5.4 *Comparative Performance of 2000 Node Client WMNs*

We simulated the performance of all the nine approaches on the timing constraints of 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, and 5.5 s. The performance results

Table 12 Comparative performance of P3PGA on 1000 node client WMNs

	Timing constraints									
Algorithm	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4	4.5	5
AODV	–	–	–	–	5	6	8	10	14	19
DSR	–	–	–	–	–	–	–	–	–	–
ACO	–	–	–	–	–	–	–	–	–	–
GA	3	2	3	2	2	1	2	0	1	0
BBO	0	0	0	0	0	0	0	0	0	0
BBBC	7	8	6	8	4	3	3	4	1	0
FA	1	0	4	2	1	1	0	0	1	0
BAT	0	0	0	0	0	0	0	0	0	0
P3PGA	9	10	7	8	8	9	7	6	3	1

"–" means failed to produce route in any of the trials

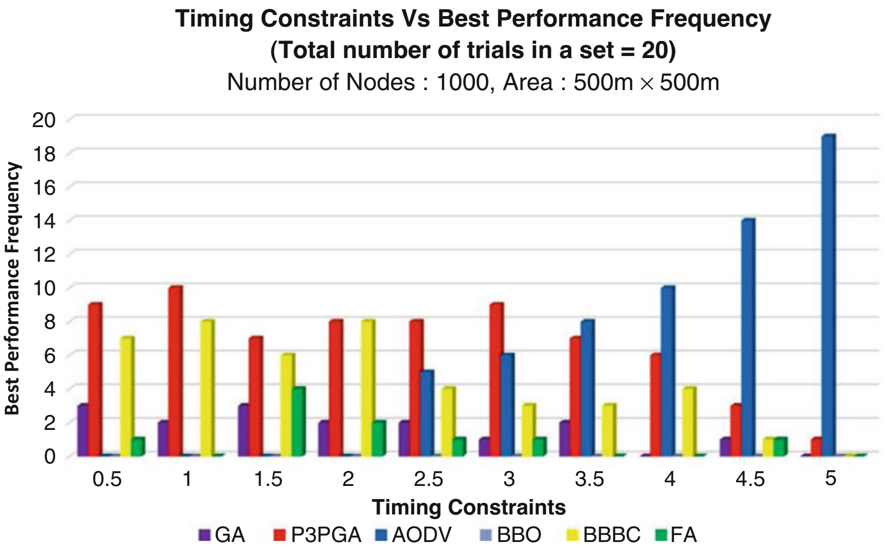


Fig. 6 Comparative performance of 1000 node client WMNs

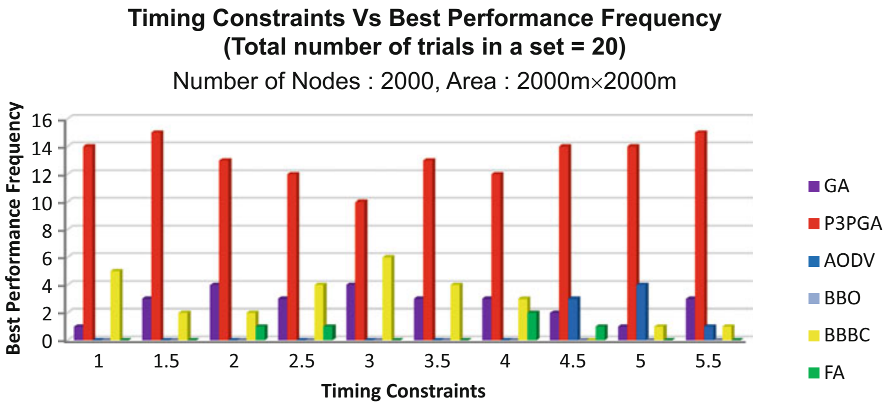


Fig. 7 Comparative unmatched best performance of 2000 node client WMNs

of all approaches are shown in Fig. 7 and Table 13. The results clearly indicate the supremacy of P3PGA approach over all other approaches for every timing constraint.

Table 13 Comparative performance of P3PGA on 2000 node client WMNs

	Timing constraints									
Algorithm	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5
AODV	–	–	–	–	–	–	0	3	4	1
DSR	–	–	–	–	–	–	–	–	–	–
ACO	–	–	–	–	–	–	–	–	–	–
GA	1	3	4	3	4	3	3	2	1	3
BBO	0	0	0	0	0	0	0	0	0	0
BBBC	5	2	2	4	6	4	3	0	1	1
FA	0	0	1	1	0	0	2	1	0	0
BAT	0	0	0	0	0	0	0	0	0	0
P3PGA	14	15	13	12	10	13	12	14	14	15

"–" means failed to produce route in any of the trials

Table 14 Comparative performance of P3PGA on 2500 node client WMNs

	Timing constraints																
Algo	2	2.5	3	3.5	4.	4.5	5.	5.5	6.	6.5	7.	7.5	8.	8.5	9.	9.5	10
AODV	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
DSR	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ACO	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
GA	3	7	6	8	3	5	6	6	2	4	3	3	4	2	3	2	2
BBO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BBBC	5	3	4	1	3	3	3	4	5	3	5	4	6	6	2	4	4
FA	0	1	2	1	0	2	0	1	1	1	1	2	2	1	0	1	2
BAT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P3PGA	12	9	8	10	14	10	11	9	12	12	11	11	8	11	15	13	12

"–" means failed to produce route in any of the trials

5.5 *Comparative Performance of 2500 Node Client WMNs*

We also evaluated the performance of all nine approaches on 2500 node client WMNs. To test the performance of all approaches we considered 17 trial sets with each set consisting of 20 trials. Here we have considered more trial sets as compared to the previous network scenarios because the network is larger and here is the need to evaluate the performance of the network on larger timing constraints also. The simulation results of all approaches are shown in Table 14 and Fig. 8. From the results we observe that the P3PGA approach outperforms all other approaches on all the timing constraints. We also observed that AODV, DSR, and ACO approaches fail to discover the route in any of the trial set. Table 15 shows that out of 340 trials, P3PGA has given the best unmatched performance 188 times, BBBC 65 times, and GA produced the best performance 69 times.

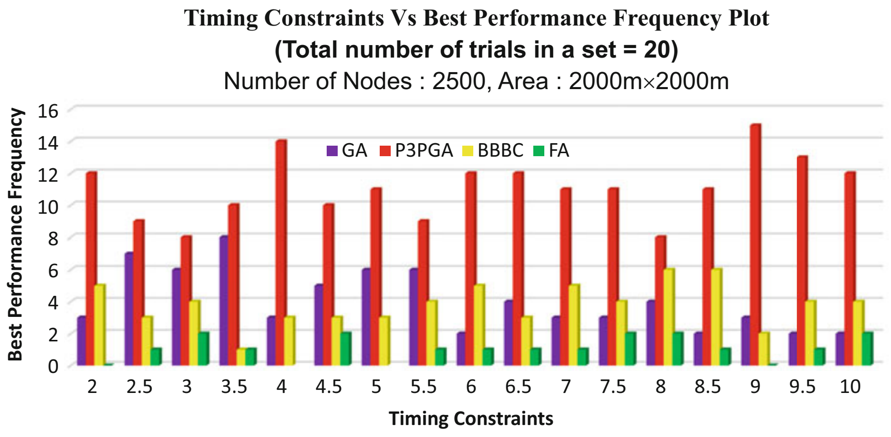


Fig. 8 Comparative performance of 2500 node client WMNs

Table 15 Overall comparative performance of P3PGA

Number of nodes	Timing constraints										
	Trials	P3PGA	FA	BBBC	BAT	AODV	BBO	GA	DSR	ACO	ALL EQUALS
100	200	41	5	24	0	37	1	14	2	0	76
500	200	49	3	15	0	114	2	7	0	0	10
1000	200	68	10	44	0	62	0	16	0	0	0
2000	200	132	5	28	0	8	0	27	0	0	0
2500	340	188	18	65	0	0	0	69	0	0	0
Total	1140	478	41	176	0	221	3	133	2	0	86

5.6 *Overall Performance Considering all Networks*

In order to evaluate the performance of all 9 approaches, overall we conducted total of 1140 trials. The overall performance of the 9 approaches are given in Fig. 9 and Table 15. From the simulation results, we observe that out of total number of 1140 trials P3PGA provided the unmatched best optimal cost route 478 times, AODV 221 times, BBBC 176 times, GA 133 times, Firefly 41 times, BBO 3 times, and DSR produced optimal cost routes only 2 times. 86 times multiple approaches produced the same best performance. Also, the ACO and BAT approaches failed to produce the optimal cost route in any of the trial sets. Figure 9 shows that as the size of the WMN becomes 1000 node P3PGA algorithm gives best performance but the margin is small. As the WMN size increases to 2000 nodes and above, P3PGA gives the best performance with a very large performance lead over its counterparts.

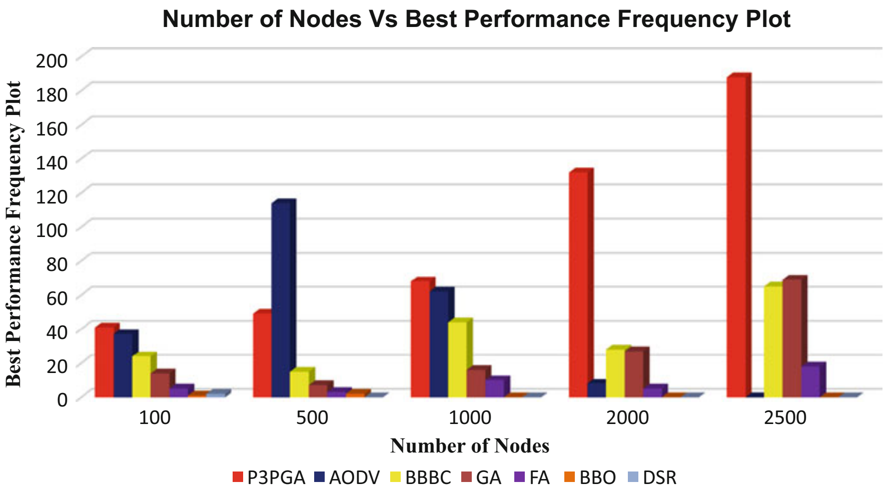


Fig. 9 Comparative performance of all approaches

6 Conclusions

This chapter proposes a new nature inspired, P3PGA-based multi-population global optimization algorithm. The proposed algorithm extended the 3PGA approach by adding the parallel evolution behavior. We implemented the proposed algorithm in MATLAB, simulated its performance on 30 benchmark functions from CEC-2014, and compared its performance with 16 other algorithms. P3PGA gave the best unmatched performance for 12 functions out of the 30 benchmark functions. On two other functions the best performance of P3PGA was equaled by some of the other algorithms. Hence, overall out of the 30 functions of CEC-2014 test suite, P3PGA gave the best performance on 14 functions. The performance of P3PGA was followed by UMOEAS, which gave unmatched best performance on one function and equaled best performance on eight functions totaling nine functions with best performance. LSHADE algorithm followed on the third place.

This chapter also proposed a P3PGA-based new optimal cost or near shortest route evaluation approach for WMNs. The approach was compared with eight other approaches, namely AODV, DSR, BBBC, ACO, BBO, BAT, GA, and Firefly-based optimal cost route evaluation approaches. From the simulation results we conclude that the proposed approach is very suitable for large WMNs with sizes greater than 1000 nodes.

The authors further suggest that the proposed P3PGA algorithm can be used in other applications such as for rule base extraction from numerical data for the fuzzy logic-based systems and for identification of fuzzy and ANN models from the given training data set.

Appendix 1: Algorithm Performance Results

Table 6 CEC-2014 Benchmark performance of various algorithms

ALGORITHM	**f1**	**f2**	**f3**	**f4**	**f5**	**f6**	**f7**	**f8**	**f9**	**f10**
NRGA	2.790E+04	9.147E+02	1.517E+03	1.544E+01	1.961E+01	2.450E+00	2.030E–01	5.585E+00	8.694E+00	1.194E+02
FWA-DM	5.013E+03	1.342E–04	0.000E+00	1.413E+00	2.003E+01	7.063E–01	9.480E–02	2.536E–01	6.008E+00	1.593E+00
UMOEAS	0.000E+00	0.000E+00	0.000E+00	0.000E+00	1.683E+01	0.000E+00	0.000E+00	0.000E+00	2.725E+00	3.739E–01
SOO + BOBYQA	4.570E+03	3.600E–02	5.843E+03	0.000E+00	2.000E+01	2.000E–03	4.900E–02	1.890E+01	8.955E+00	1.304E+02
SOO	8.811E+06	6.643E+00	6.644E+03	6.780E–01	2.000E+01	2.000E–03	4.900E–02	1.890E+01	8.955E+00	1.304E+02
RSDE	0.000E+00	0.000E+00	0.000E+00	2.811E+00	1.922E+01	5.291E–02	3.550E–02	6.608E–01	8.522E+00	6.844E+01
POBL_ADE	1.620E+04	2.270E+03	5.740E–04	2.550E+01	1.910E+01	1.040E+00	1.630E–01	7.810E+00	7.630E+00	1.530E+02
FERDE	2.368E+00	6.288E–05	1.346E–03	0.000E+00	1.906E+01	8.890E–01	1.883E–02	0.000E+00	5.638E+00	3.674E–02
FCDE	0.000E+00	0.000E+00	0.000E+00	1.841E+01	2.033E+01	3.566E+00	1.961E–01	1.607E+01	2.099E+01	2.919E+02
DE_b6e6rlwithrestart	0.000E+00	0.000E+00	0.000E+00	1.125E+00	1.845E+01	0.000E+00	1.688E–02	0.000E+00	4.895E+00	1.225E–03
CMLSP	1.769E–07	0.000E+00	1.056E–04	0.000E+00	1.686E+01	6.201E–02	0.000E+00	2.071E+00	1.659E+00	1.961E+02
GaAPADE	0.000E+00	0.000E+00	0.000E+00	3.069E+01	1.968E+01	1.484E–01	3.163E–03	0.000E+00	3.379E+00	1.518E–01
OptBees	7.842E+02	9.883E–03	9.213E–01	2.691E+00	2.000E+01	3.017E+00	1.562E–01	0.000E+00	2.084E+01	2.192E+02
LSHADE	0.000E+00	0.000E+00	0.000E+00	2.941E+01	1.415E+01	1.754E–02	3.043E–03	0.000E+00	2.345E+00	8.572E–03
RMA-LSCh-CMA	0.000E+00	0.000E+00	1.025E–07	8.501E–02	1.365E+01	1.479E–04	0.000E+00	0.000E+00	3.317E+00	7.678E+00
MVMO	4.954E–04	0.000E+00	0.000E+00	9.546E+00	1.658E+01	3.445E–03	1.858E–02	0.000E+00	3.492E+00	2.137E+00
P3PGA	**1.76E+01**	**1.480E+01**	**0.000E+00**	**5.716E–02**	**0.000E+00**	**7.236E–01**	**9.855E–02**	**0.000E+00**	**1.161E+00**	**0.000E+00**

(continued)

Table 6 (continued

ALGORITHM	**f11**	**f12**	**f13**	**f14**	**f15**	**f16**	**f17**	**f18**	**f19**	**f20**
NRGA	5.759E+02	1.242E–01	1.577E–01	2.537E–01	1.022E+00	2.747E+00	1.607E+04	7.420E+03	2.093E+00	1.719E+03
FWA-DM	3.722E+02	4.249E–02	1.206E–01	2.139E–01	7.748E–01	1.757E+00	2.545E+02	2.516E+01	1.299E+00	1.337E+01
UMOEAS	1.440E+02	0.000E+00	9.436E–03	1.100E–01	6.667E–01	1.530E+00	8.477E+00	7.840E–01	2.000E–01	3.706E–01
SOO + BOBYQA	3.491E+02	0.000E+00	3.000E–02	1.300E–01	4.200E–01	2.520E+00	4.226E+02	3.952E+03	5.500E–01	6.925E+03
SOO	3.491E+02	0.000E+00	3.000E–02	1.300E–01	4.400E–01	2.520E+00	3.123E+06	1.293E+04	5.500E–01	9.364E+03
RSDE	2.906E+02	2.206E–01	1.277E–01	1.360E–01	9.830E–01	2.233E+00	4.770E+01	1.996E+00	1.030E+00	7.215E–01
POBLADE	2.080E+02	2.690E–01	1.310E–01	2.600E–01	7.120E–01	1.410E+00	2.570E+02	3.320E+01	2.090E+00	1.260E+01
FERDE	7.554E+01	1.227E–01	1.158E–01	9.359E–02	6.725E–01	1.530E+00	8.230 + 00	2.730E+00	5.092E–01	1.704E+00
FCDE	7.554E+01	1.227E–01	1.158E–01	9.359E–02	6.725E–01	1.530E+00	8.230E+00	2.730E+00	5.092E–01	1.704E+00
DE_b6e6rlwithrestart	1.965E+02	2.929E–01	1.281E–01	1.113E–01	8.317E–01	1.872E+00	1.398E+00	6.207E–01	1.418E–01	5.593E–02
CMLSP	1.530E+02	3.027E–02	2.725E–02	1.892E–01	8.966E–01	1.555E+00	3.127E+02	3.085E+01	1.251E+00	1.994E+01
GaAPADE	1.831E+02	1.402E–01	6.009E–02	9.424E–02	6.057E–01	1.977E+00	9.914E+00	2.230E–01	2.566E–01	4.316E–01
OptBees	3.927E+02	1.304E–01	4.162E–01	3.687E–01	2.439E+00	2.640E+00	6.844E+02	3.350E+01	9.330–01	8.958E+00
LSHADE	3.206E+01	6.817E–02	5.156E–02	8.136E–02	3.661E–01	1.241E+00	9.767E–01	2.441E–01	7.730E–02	1.849E–01
RMA-LSCh-CMA	2.013E+01	1.646E–02	3.292E–02	1.265E–01	4.715E–01	1.054E+00	7.834E+01	5.221E+00	7.661E–02	8.057E+00
MVMO	9.628E+01	4.223E–02	3.553E–02	8.906E–02	4.346E–01	1.449E+00	9.357E+00	7.826E–01	1.583E–01	3.126E–01
P3PGA	**3.555E+00**	**1.365E–06**	**3.876E–02**	**2.537E–02**	**3.294E–01**	**1.662E–01**	**4.824E+01**	**6.326E+00**	**4.892E–02**	**7.684E–02**

ALGORITHM	**f21**	**f22**	**f23**	**f24**	**f25**	**f26**	**f27**	**f28**	**f29**	**f30**
NRGA	4823.427182	37.56658082	329.4574872	130.7641476	183.6782269	100.1366	280.7775666	477.1473826	413.2909874	1727.537695
FWA-DM	9.464E+01	3.409E+01	3.295E+02	1.274E+02	1.787E+02	100.1384	3.213E+02	3.472E+02	2.117E+02	3.943E+02
UMOEAS	5.404E–01	2.448E–01	3.295E+02	1.083E+02	1.260E+02	100.0140	2.548E+01	3.129E+02	1.955E+02	2.339E+02
SOO + BOBYQA	1.940E+03	1.265E+02	2.00E+02	1.157E+02	1.391E+02	100.0500	2.000E+02	2.000E+02	2.000E+02	2.000E+02
SOO	2.469E+04	1.265E+02	2.000E+02	1.157E+02	1.452E+02	100.0500	2.000E+02	2.000E+02	2.000E+02	2.000E+02
RSDE	1.209E+00	1.165E+01	3.295E+02	1.191E+02	1.295E+02	100.1291	9.125E+01	3.869E+02	2.126E+02	5.052E+02
POBL_ADE	1.030E+02	3.000E+01	3.290E+02	1.240E+02	1.860E+02	100.0000	2.560E+02	4.230E+02	3.550E+05	6.380E+02
FERDE	8.543E+00	3.242E+00	3.295E+02	1.146E+02	1.363E+02	100.0901	3.664E+02	3.664E+02	3.182E+02	5.348E+02
FCDE	1.481E+02	2.750E+01	3.295E+02	1.369E+02	1.840E+02	100.3461	4.752E+01	4.569E+02	3.405E+04	8.667E+02
DE_b6e6rlwith restart	7.867E–01	1.541E–01	3.295E+02	1.122E+02	1.290E+02	100.1170	6.161E+01	3.634E+02	2.178E+02	4.673E+02
CMLSP	3.639E+01	8.953E+01	2.018E+02	1.099E+02	1.275E+02	100.0194	4.113E+01	2.803E+02	2.000E+02	2.164E+02
GaAPADE	5.086E–01	3.247E+00	3.295E+02	1.089E+02	1.636E+02	100.0688	8.969E+01	3.832E+02	2.223E+02	4.672E+02
OptBees	5.706E+01	1.702E+01	2.724E+02	1.374E+02	1.460E+02	100.3964	7.423E+00	3.067E+02	2.200E+02	3.892E+02
LSHADE	4.081E–01	4.410E–02	3.295E+02	1.075E+02	1.327E+02	100.0500	5.806E+01	3.808E+02	2.220E+02	4.649E+02
RMA-LSCh-CMA	4.929E+01	8.475E+00	3.295E+02	1.084E+02	1.751E+02	100.0364	1.848E+02	3.887E+02	2.271E+02	5.851E+02
MVMO	1.935E+00	2.629E–01	3.295E+02	1.092E+02	1.161E+02	100.0323	1.720E+01	3.611E+02	1.814E+02	4.917E+02
P3PGA	**0.212959783**	**0.119165579**	**329.4574747**	**106.9715459**	**113.0468507**	**100.0354**	**1.26866855**	**356.3869066**	**203.2702669**	**492.4918411**

References

1. D. Goldberg, *Genetic Algorithms in Optimization, Search and Machine Learning* (Addison-Wesley, Reading, 1989)
2. B.S. Khera, P.A.P. Singh, Comparison of genetic algorithm, particle swarm optimization and biogeography-based optimization for feature selection to classify clusters of micro calcifications. J. Inst. Eng. (India): Series B **98**(2), 189–202 (2017)
3. S. Suresh Optimized scheme for grid computations using genetic algorithms, *Proceedings of the International Conference on Internet Technologies & Applications*, Wrexham, UK, September 4–7, 2007
4. M. Melanie, S. Forrest, Genetic algorithms and artificial life. Artif. Life **1**(3), 267–289 (1994)
5. J.H. Holland, *Adaptation in Natural and Artificial Systems, Ph.D. Thesis* (University of Michigan Press, Ann Arbor, MI, 1975)
6. H. Mühlenbein and H. M. Voigt, Gene pool recombination in genetic algorithms, in Meta-Heuristics: Theory and Applications, Springer US, pp. 53–62 (1996)
7. A. Eiben, C.H. Van Kemenade, Diagonal crossover in genetic algorithms for numerical optimization. Control. Cybern. **26**(3), 447–465 (1997)
8. A. Wu, P.W.M. Tsang, T.Y. Yuen, L.F. Yeung, Affine invariant object shape matching using genetic algorithm with multi-parent orthogonal recombination and migrant principle. Appl. Soft Comput. **9**(1), 282–289 (2009)
9. A.E. Eiben, P.E. Raue, and Z. Ruttkay, Genetic algorithms with multi-parent recombination, in *International Conference on Evolutionary Computation The Third Conference on Parallel Problem Solving from Nature Jerusalem, Israel*, p. 78–87 (1994)
10. P. Amato, M. Tachibana, M. Sparman, S. Mitalipov, Three-parent in vitro fertilization: Gene replacement for the prevention of inherited mitochondrial diseases. Fertil. Steril. **101**(1), 31–35 (2014)
11. H. Fertilisation and E. Authority, (2014) Third scientific review of the safety and efficacy of methods to avoid mitochondrial disease through assisted conception: 2014 update
12. J. Hamzelou, Everything you wanted to know about '3- parent' babies. [Online] (2016). Available: https://www.newscientist.com/article/2107451-everything-you-wanted-to-know-about-3-parent-babies/
13. J. Hamzelou, Exclusive: Worlds first baby born with new 3 parent technique. [Online] (2016). Available: https://www.newscientist.com/article/2107219-exclusive-worlds-first-baby-born-with-new-3-parent-technique/
14. I.F. Akyildiz, X. Wang, W. Wang, Wireless mesh networks: A survey. Comput. Netw. **47**(4), 445–487 (2005)
15. S. Amar, *Some Nature Inspired Computing Approaches to Routing in Wireless Mesh Networks, Ph.D. Thesis* (Submitted to IKG Punjab Technical University, Jalandhar (India), 2017)
16. S. Amar, K. Shakti, S. Ajay, S.S. Walia, Three-parent GA: A global optimization algorithm. J. Mult. Valued Log. Soft Comput. **32**, 407–423 (2019)
17. T. Blickle and L. Thiele, A comparison of selection schemes used in genetic algorithms, TIK Report No. 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, (1995)
18. J.E. Baker Adaptive selection methods for genetic algorithms, in *Proceedings of International Conference on Genetic Algorithms and their applications*, p. 101–111 (1985)
19. J.E. Baker, Reducing bias and inefficiency in the selection algorithm. in Proceedings of the Second International Conference on Genetic Algorithms, Vol. 206, p. 14–21 (1987)
20. S.M. Elsayed, R.A. Sarker, D.L. Essam and N.M. Hamza, Testing united multi-operator evolutionary algorithms on the CEC2014 real-parameter numerical optimization, IEEE Congress on Evolutionary Computation (CEC), IEEE, p. 1650–1657 (2014)
21. R. Tanabe and A.S. Fukunaga, (2014) Improving the search performance of SHADE using linear population size reduction, IEEE Congress on Evolutionary Computation (CEC), p. 1658–1665

22. C. Xu, H. Huang and S. Ye, A differential evolution with replacement strategy for real-parameter numerical optimization. IEEE Congress on Evolutionary Computation (CEC), p. 1617–1624 (2014)
23. B.Y. Qu, J.J. Liang, J.M. Xiao and Z.G. Shang, Memetic differential evolution based on fitness Euclidean-distance ratio, IEEE Congress on Evolutionary Computation (CEC), p. 2266–2273 (2014)
24. Z. Hu, Y. Bao and T. Xiong, Partial opposition-based adaptive differential evolution algorithms: evaluation on the CEC 2014 benchmark set for real-parameter optimization", IEEE Congress on Evolutionary Computation (CEC), pp. 2259–2265 (2014)
25. Z. Li, Z. Shang, B.Y. Qu and J.J. Liang, Differential evolution strategy based on the constraint of fitness values classification, IEEE Congress on Evolutionary Computation (CEC), p. 1454–1460 (2014)
26. I. Erlich, J.L. Rueda, S. Wildenhues and F. Shewarega, Evaluating the mean-variance mapping optimization on the IEEE-CEC 2014 test suite, IEEE Congress on Evolutionary Computation (CEC), p. 1625–1632 (2014)
27. D. Molina, B. Lacroix and F. Herrera, Influence of regions on the memetic algorithm for the CEC'2014 Special Session on real-parameter single objective optimization, IEEE Congress on Evolutionary Computation (CEC), p. 1633–1640 (2014)
28. R.D. Maia, L.N. de Castro and W.M. Caminhas, Real-parameter optimization with OptBees, IEEE Congress on Evolutionary Computation (CEC), p. 2649–2655 (2014)
29. P. Preux, R. Munos and M. Valko, Bandits attack function optimization, IEEE Congress on Evolutionary Computation (CEC) (2014)
30. C. Yu, L. Kelley, S. Zheng and Y. Tan, Fireworks algorithm with differential mutation for solving the CEC 2014 competition problems, IEEE Congress on Evolutionary Computation (CEC), p. 3238–3245 (2014)
31. L. Chen, Z. Zheng, H.L. Liu and S. Xie An evolutionary algorithm based on covariance matrix leaning and searching preference for solving CEC 2014 benchmark problems, IEEE Congress on Evolutionary Computation (CEC), p. 2672–2677 (2014)
32. R. Mallipeddi, G. Wu, M. Lee and P.N. Suganthan, Gaussian adaptation based parameter adaptation for differential evolution, IEEE Congress on Evolutionary Computation (CEC), p. 1760–1767 (2014)
33. D. Yashesh, K. Deb and S. Bandaru, Non-uniform mapping in real-coded genetic algorithms, IEEE Congress on Evolutionary Computation (CEC), p. 2237–2244 (2014)
34. R. Poláková, J. Tvrdík and P. Bujok, Controlled restart in differential evolution applied to CEC 2014 benchmark functions. IEEE Congress on Evolutionary Computation (CEC), p. 2230–2236 (2014)
35. A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou, A multi radio communication protocol for IEEE 802.11 wireless networks, Proceedings of International Conference on Broadcast Networks (Broad Nets), San Jose, California, USA, October 25–29, p. 344–354 (2004)
36. R. Draves, J. Padhye, and B. Zill, Comparisons of routing metrics for static multi-hop wireless networks, Proceedings of ACM Annual Conference of the Special Interest Group on Data Communication (SIGCOMM), Portland, Oregon, USA, August 30–September 03, p. 133–144 (2004)
37. D.S.J. DeCouto, D. Aguayo, J. Bicket, R. Morris, A high throughput path metric for multihop wireless routing, Proceedings of ACM Annual International Conference on Mobile Computing and Networking (MOBICOM), San Diego, CA, USA, September 14–19, p. 134–146 (2003)
38. R. Draves, J. Padhye, and B. Zill, Routing in multi-radio, multihop wireless mesh networks, Proceedings of ACM annual International conference on mobile computing and networking (Mobi Con04), Philadelphia, Pennsylvania, USA, September 26–October 01, p. 114–128 (2004)
39. G. Jakllari, S. Eidenbenz, N. Hengartner, S. Krishnamurthy, and M. Faloutsos, Link positions matter: A noncommutative routing metric for wireless mesh networks, Proceedings of IEEE Annual Conference on Computer Communications (INFOCOM), Phoenix, Arizona, USA, April 13–18, p. 744–752 (2008)

40. C.E. Koksal, and H. Balakrishnan, Quality-aware routing metrics for time varying wireless mesh networks, IEEE Journal on Selected Areas in Communications, 24(11), p. 1984–1994 (2006)
41. Y. Yang, J. Wang, R. Kravets, Interference-aware load balancing for multi hop wireless networks, Technical Report UIUCDCSR-2005-2526, University of Illinois at Urbana Champaign, Department of Computer Science, and Web Address: http://www.ideals.uiuc.edu/handle/2142/10974, (2005)
42. T. Liu and W. Liao, Capacity-aware routing in multi-channel multi-rate wireless mesh networks, Proceedings of IEEE International Conference on Communications (ICC), Istanbul, Turkey, 11–15 June, p. 1971–1976 (2006)
43. G. Karbaschi and A. Fladenmuller, A link quality and congestion-aware cross layer metric for multi-hop wireless routing, Proceedings of IEEE International Conference on Mobile Ad hoc and Sensor Systems Conference, Washington, DC, USA, 2005, 7 Nov. 2005, p. 7–11
44. L. Ma, Q. Zhang, Y. Xiong, and W. Zhu, Interference aware metric for dense multi-hop wireless network, Proceedings of IEEE International Conference on Communications (ICC), Seoul, South Korea, pp. 1261–1265 (2005)
45. S. Sharma, S. Kumar, B. Singh, Routing in wireless mesh networks: Three new nature inspired approaches. Wirel. Pers. Commun. **83**(4), 3157–3179 (2015)
46. S. Yang, H. Cheng, F. Wang, Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **40**(1), 52–63 (2010)

Application of Evolutionary Algorithms to Power System Stabilizer Design

Tshina Fa Mulumba and Komla Agbenyo Folly

1 Introduction

1.1 Oscillations in Electrical Power Systems and Power Systems Stabilizers

Low-frequency oscillations in the range of 0.2–3 Hz are inherent to power systems designed for the supply, transfer, and utilization of electrical power [1]. They appear when there are power exchanges between large areas of interconnected power systems or when power is transferred over long distances under medium to heavy conditions.

Since the development of interconnected power systems and the introduction of deregulation of electrical power systems, these oscillations commonly known as electromechanical modes have become apparent especially during and after small and large disturbances [2–4]. Several factors contribute to the rise of these oscillations. The use of high-gain fast-acting automatic voltage regulator (AVR), necessary to increase the ability of the system to maintain the stability during faults, has adverse effects on the system damping due to the introduction of a lagging phase angle between the input voltage reference and the electrical output power [1, 2, 4]. Moreover, the recent exponential increase in power demand which has led to bulk power transfer over weak transmission lines has also been found to cause oscillations that can limit the maximum transfer capability of the system. If no adequate damping is provided, these oscillations could grow in magnitude with time and lead to system collapse [1, 2, 4–6].

T. F. Mulumba · K. A. Folly (✉)
Department of Electrical Engineering, University of Cape Town, Cape Town, South Africa
e-mail: komla.folly@uct.ac.za

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_2

To mitigate these oscillations, various controllers have been developed and implemented over the years. Power systems stabilizers (PSSs) have been extensively used as supplementary excitation controllers that provide additional damping to eliminate electromechanical oscillations and enhance the overall system stability. To achieve this, the conventional PSS (CPSS) is often designed at a particular operating condition using conventional methods such as phase compensation and root locus. However, due to the non-linearity characteristics of power systems and the varying operating conditions, the performance of the CPSS deteriorates as the operating conditions change and therefore require re-tuning [7–13].

In recent years, increasing interest has been focused on the optimization of stabilizers parameters to provide adequate performance for a wide range of operating conditions [14–20]. Consequently, many conventional optimization techniques and computational intelligence (CI) techniques, namely, evolutionary algorithms (EAs) such as genetic algorithms (GAs) [21], differential evolution (DE) [22–31], population-based incremental learning (PBIL) [32–37], etc., have been used to find an optimal set of parameters that guarantee robust performance under varying operating conditions. However, the performances of these algorithms greatly depend on their parameters, and therefore could suffer from the problem of being trapped in local optima. In this chapter, we investigated the use of adaptive or self-adaptive parameter schemes of these algorithms (mainly DE and PBIL) to achieve improved performances.

1.2 Algorithms for Parameter Optimization: Differential Evolution and Population-Based Incremental Learning

Evolutionary algorithms (EAs) are population-based optimizers inspired by the mechanism of evolution and natural selection. They solve the optimization problem by sampling the objective function at multiple random initial points in the search space, and explore the search space by iteratively generating new points that are perturbation of the existing ones [22]. This approach is convenient in locating the global maximum/minimum instead of local. EAs are simple, robust, efficient, and versatile algorithms that can be applied to a wide variety of problems, including those involving non-linear, discontinuous, or complex functions. In the last few years, increasing number of researchers have proposed EAs to optimally tune the parameters of the PSS to guarantee a robust performance over a wide range of system conditions.

A particular strain of EAs known as genetic algorithms (GAs) has received particular attention in the last few decades. GAs are heuristic population-based search methods inspired by the mechanism of biological genetic evolution. GAs have been used to design PSSs for multimachines system [7–9, 38].

Despite GAs' performance and promising results in numerous applications, recent analyses have revealed some drawbacks [39]. The problem of genetic drift in GAs restricts the population diversity and the search space for solutions. As a result, GAs may converge to suboptimal solutions [7–9, 39]. In addition, GAs are expensive in terms of both computation time and memory when dealing with difficult problems such as tuning PSSs in a multimachine environment [7, 8, 40].

To cope with the above drawbacks, many variants of GAs have been proposed, which are often tailored to a particular problem. Recently, several simpler and yet effective heuristic algorithms have received increasing attention. They have shown their potential in global optimization problems to overcome the deficiencies of GAs in exploring wider spaces for the global maxima [22]; this chapter focuses on the application of two of these variants: differential evolution (DE) and population-based incremental learning (PBIL). These two optimization algorithms are introduced briefly below.

Differential Evolution (DE) is a powerful stochastic optimizer whose search mechanism involves a differential mutation technique. The algorithm is both simple and robust, with several variants exhibiting different trade-offs between convergence speed and robustness.

DE has been the subject of intensive performance evaluation since its inception. Many comparisons have been carried out with other optimization algorithms on benchmark functions and several other applications. Most often DE outperforms its counterparts in efficiency and robustness [41].

DE has been increasingly applied to a variety of problems, especially in engineering [41]. In Wang et al. [42], DE was successfully applied in designing PSS. The resulting PSS was then compared to the CPSS through a series of tests: the DE-based PSS outperformed the CPSS. In Mulumba and Folly [43], DE is compared to GAs when used to simultaneously tune PSSs. The results have proven that DE outperformed GAs.

One problematic issue in the use of DE is the choice of algorithm parameters, which may have a large effect on algorithm performance. To address this issue, self-adaptive schemes have been developed whereby the mutation and crossover parameters are changed during the DE optimization process. Some of these are discussed in Sect. 3.3.

Population-Based Incremental Learning (PBIL): is a method that combines genetic algorithms and competitive learning for function optimization. PBIL is an extension to the Evolutionary Genetic Algorithm (EGA) algorithm achieved through the re-examination of the performance of the EGA in terms of competitive learning [32]. In Sheetekela and Folly [7–9, 44], PBIL was used to tune PSS parameters. The results showed considerable enhancement in PSS performance compared to PSS obtained with GAs. One disadvantage of PBIL is the slow convergence due to the learning process involved. This disadvantage can be addressed by optimally tuning the learning rate (α) parameter by trial and error. Even when α is tuned, PBIL still requires a large number of generations to obtain a solution.

Other methods that have been used to design PSS are particle swarm optimization (PSO) [45], and simulated annealing (SA) [46]. These are not discussed in this chapter.

2 Problem Statement

2.1 Overview

Small signal stability refers to the ability of a power system to maintain stability when subject to small disturbances [1]. The disturbances are considered small only if linearization of the system is possible. Hence, this chapter reviews linear techniques used to analyse small signal oscillations and extract information about the system dynamic characteristic. Techniques such as modal analysis, eigenvectors, eigenvalues' sensitivity, and participation factors are touched on in subsequent sections.

2.2 State-Space Representation

The state-space representation is often used to describe the behaviour of a dynamical system. The state-space representation employs a set of n first-order non-linear differential equations to model the system, which can be written in vector form as:

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}; \mathbf{u}; t\right) \tag{1}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

$\mathbf{x}$: the state vector containing the variables which describe the state of the system (state variables);
$\dot{\mathbf{x}}$: derivative vector of the state variables;
$\mathbf{f}$: vector of non-linear functions;
$\mathbf{u}$: vector of the inputs;
t: time;
n: the order of the system;
r: the number of inputs.

If the vector function $\mathbf{f}$ is not an explicit function of time the system is referred to as *autonomous*, and (1) simplifies to:

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}; \mathbf{u}\right) \tag{2}$$

The equation relating the system outputs to the system inputs and state variables can be written as

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}) \tag{3}$$

where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} g_1 \\ \vdots \\ g_m \end{bmatrix}$$

y: vector of system outputs;
g: vector of non-linear functions relating the state and input variables to the output variables.
m: number of outputs.

2.3 Linearization

An *equilibrium point* of an autonomous system is a vector pair $(\mathbf{x_0}, \mathbf{u_0})$ which satisfies the following equilibrium condition:

$$\mathbf{f}(\mathbf{x_0}, \mathbf{u_0}) = 0 \tag{4}$$

In view of (2), Eq. (4) implies that $\dot{\mathbf{x}}_\mathbf{0} = \mathbf{0}$, so that $\mathbf{x_0}$ is a time-independent solution to the state equation when the system inputs are given by $\mathbf{u_0}$.

Following a perturbation, $\mathbf{\Delta x}$ and $\mathbf{\Delta u}$, in the system state and input variables, we get

$$\dot{\mathbf{x}} = \dot{\mathbf{x_0}} + \Delta\dot{\mathbf{x}} \quad \mathbf{u} = \mathbf{u_0} + \Delta\mathbf{u}, \tag{5}$$

and (2) may be rewritten as:

$$\dot{\mathbf{x}} = \dot{\mathbf{x}}_\mathbf{0} + \Delta\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x_0} + \Delta\mathbf{x}, \mathbf{u_0} + \Delta\mathbf{u}) \tag{6}$$

If the perturbations in the system are sufficiently small, the right-hand side of Eq. (6) may be closely approximated by a first-order Taylor series expansion in $\Delta\mathbf{x}$ and $\Delta\mathbf{u}$. After algebraic simplification (and making use of Eq. (4), we obtain)

$$\Delta\dot{x}_i = \frac{\partial f_i}{\partial x_1}\Delta x_1 + \cdots + \frac{\partial f_i}{\partial x_n}\Delta x_n + \frac{\partial f_i}{\partial u_1}\Delta u_1 + \cdots + \frac{\partial f_i}{\partial u_r}\Delta u_r \tag{7}$$

where $i = 1, 2, 3, \ldots, n$ and

$$\Delta y_j = \frac{\partial g_j}{\partial x_1}\Delta x_1 + \cdots + \frac{\partial g_j}{\partial x_n}\Delta x_n + \frac{\partial g_j}{\partial u_1}\Delta u_1 + \cdots + \frac{\partial g_j}{\partial u_r}\Delta u_r \tag{8}$$

where $j = 1, 2, 3, \ldots, m$.

Equations (7) and (8) may be rewritten in matrix form as:

$$\Delta\dot{\mathbf{x}} = \mathbf{A}\Delta\mathbf{x} + \mathbf{B}\Delta\mathbf{u} \tag{9}$$

$$\Delta\mathbf{y} = \mathbf{C}\Delta\mathbf{x} + \mathbf{D}\Delta\mathbf{u} \tag{10}$$

where:

$$\mathbf{A} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial \mathrm{x}_n} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_r} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_r} \end{bmatrix} \tag{11}$$

$$\mathbf{C} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \cdots & \frac{\partial g_m}{\partial x_n} \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} \frac{\partial g_1}{\partial u_1} & \cdots & \frac{\partial g_1}{\partial u_r} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_m}{\partial u_1} & \cdots & \frac{\partial g_m}{\partial u_r} \end{bmatrix}$$

$\mathbf{\Delta x}$ is the linearized state vector of dimension n;
$\mathbf{\Delta y}$ is the linearized output vector of dimension m;
$\mathbf{\Delta u}$ is the linearized input vector of dimension r;
$\mathbf{A}$ is the state matrix of size $n \times n$;
$\mathbf{B}$ is the input matrix, size $n \times r$;
$\mathbf{C}$ is the output matrix, size $m \times n$;
$\mathbf{D}$ is the feed forward matrix, size $m \times r$;

After linearization, the system stability can be analysed using the modal analysis as follows.

2.4 Modal Analysis

Once the state space has been established, the stability of the system can be extracted and analysed by means of eigenvalues, eigenvector properties, and the participation factor. These terms are defined and explained below.

Eigenvalues

Given an $n \times n$ matrix $\mathbf{A}$, the *eigenvectors* and *eigenvalues* of $\mathbf{A}$ may be defined in terms of the following equation:

$$\mathbf{A\Phi} = \lambda\mathbf{\Phi}, \tag{12}$$

where $\mathbf{\Phi}$ is a $n \times l$ vector of complex numbers, and λ is a complex number. Any non-zero vector $\mathbf{\Phi}$ that satisfies (12) is called an *eigenvector* of $\mathbf{A}$: and the corresponding value of λ is called the *eigenvalue.*

By rearranging Eq. (12), we may show that any eigenvalue λ of $\mathbf{A}$ must satisfy the condition:

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0 \tag{13}$$

where $\mathbf{I}$ is the identity matrix. Using linear algebra, it can also be shown that (13) is also a sufficient condition for eigenvalues: in other words, for every λ that satisfies (13) there exist non-zero vectors $\mathbf{\Phi}$ that satisfy (12).

Equation (13) is a polynomial equation of order n, which implies there are at most n roots which we denote as $(\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_n)$. Since these eigenvalues may be complex, we may write $\lambda_k = \sigma_k \pm j\omega_k$, where σ_k and ω_k are real. If $\mathbf{A}$ has real entries, then the complex eigenvalues occurs in conjugate pairs.

The stability of the system at an operating point (x_0, u_0) can be determined by analysing the eigenvalues. Real eigenvalues correspond to non-oscillatory modes, and eigenvalues with non-zero imaginary parts are oscillatory mode. If the real eigenvalue is negative, the mode decays over time, and is called stable; and if positive, the mode is said to have an aperiodic instability. For conjugate eigenvalue pairs $(\sigma \pm j\omega)$, if the real component σ is negative, then the mode is oscillatory stable; otherwise it is oscillatory unstable. The imaginary component, ω, gives the oscillatory frequency in rad/s: the corresponding frequency in hertz is given by

$$f = \frac{\omega}{2\pi} \tag{14}$$

A system is said to be *stable* at a particular operating point if all modes are stable, which is equivalent to saying that all eigenvalues have negative real parts. Otherwise, the system is *unstable*.

The *damping ratio* ζ is given by the following equation:

$$\zeta = \frac{-\sigma}{\sqrt{\sigma^2 + \omega^2}} \tag{15}$$

When $\sigma < 0$, the damping ratio indicates the rate of decay of the amplitude of the oscillations. For power systems, damping ratios of 0.05 and above are considered adequate, but damping ratios of 0.2 and above are often preferred, especially for electromechanical oscillations [4, 44]. The stability of the system is determined by the eigenvalue with the smallest damping ratio.

In the current research, we are interested in optimizing the guaranteed stability of the system under several operating conditions. Each set of operating conditions will have its own eigenvalues and damping ratios. The overall system stability is determined by the eigenvalue with smallest damping ratio from among all operating conditions. It follows that the objective function to be maximized may be formulated as follows [44]:

$$ObjFn = \min\left(\zeta_{i,j}\right), \quad \text{where } i = 1, 2, \ldots n \text{ and } j = 1, 2, \ldots, m \tag{16}$$

where $\zeta_{i,j}$ is the damping ratio of the *i*th eigenvalue for the '*j*th' operating condition. In terms of *ObjFn*, the optimization problem may be phrased as follows:

$$\text{Maximize} \quad ObjFn \tag{17}$$

Subject to:

$$K_p^{\min} \le K_p \le K_p^{\max} \quad \text{where } p = 1, 2, \ldots, m$$

$$T_{q,p}^{\min} \le T_{q,p} \le T_{q,p}^{\max} \quad \text{where } q = 1, 2, \ldots, n \text{ and } p = 1, \ldots, m,$$

where *K* and *T* denote the optimized controller gain and the lead-lag time constants, respectively, within their respective boundaries. The subscript '*q*' indexes the lead-lag parameter number, while '*p*' is the machine number. In this chapter the following values were used:

$$K_p^{\max} = 20; \quad T_{q,p}^{\max} = 5 \text{ s}$$

$$K_p^{\min} = 1; \quad T_{q,p}^{\min} = 0.001 \text{ s}.$$

The objective function was applied to various system configurations, such as single machine infinite bus and two-area multi-machine systems. These systems are described below.

3 The Differential Evolution Algorithm

This section provides an introduction to the differential evolution technique. A detailed description of the searching mechanism used to find the optimum value of functions is presented below.

3.1 Overview

Differential evolution (DE) was originally developed by Price and Storn in 1995 in an effort to overcome Genetic algorithms' shortcomings in solving the Chebyshev polynomial fitting problem [22, 47]. DE is designed to efficiently solve non-differentiable and non-linear functions while retaining a simple algorithmic procedure and good convergence to a global optimum. DE can be characterized as a parallel direct search method that uses a population of points to search for a global minimum or maximum of a function over a wide search space. Similar to most EAs, DE simulates the Darwinian evolution theory to direct its search towards prospective areas. However, unlike traditional genetic algorithms (GAs) which uses "bit-string" encoding, DE encodes all parameters as floating-point regardless of their type. The floating-point representation offers efficient memory utilization, and its one-to-one selection strategy lowers computational complexity which subsequently increases the speed of convergence [22].

The DE algorithm starts by sampling the search space at various randomly chosen initial points, and then generates new points that are perturbations (or mutations) of existing points. The perturbation is achieved by adding the scaled difference between two randomly selected vectors to a third. The resultant mutation vector is crossed over with the corresponding parent to generate a trial or offspring vector [22, 47–49]. Finally, the offspring competes against its parent in a one-to-one selection process based on their fitness value. The one with the better fitness value survives and enters the next generation. The procedure is repeated for each point (or vector) in the search space to form the new generation in the evolutionary cycle. The search stops when either the solution converges to the true optimum or following a termination criterion such as when a maximum generation number is reached [8].

3.2 Detailed DE Algorithm Description

In this section a more detailed description of the algorithm is provided, following the descriptions in Storn and Price, and Price and Storn [22, 50].

3.2.1 Population Structure

In DE, the population is composed of N_p candidate solutions or points, where each candidate is a real D-dimensional vector where D is the number of parameters to be optimized. The population iterates through multiple generations. We may denote the population at the gth generation as $P_{x,g}$, where

$$P_{x,g} = \left(X_{i,g}\right), \quad i = 1, \ldots, Np, g = 0, 1, \ldots, g_{\max}. \tag{18}$$

Here $X_{i,g}$ is the ith candidate solution in the gth generation, and may be denoted as:

$$X_{i,g} = \left(x_{j,i,g}\right), \quad j = 0, 1, \ldots, D - 1,$$

so that each candidate solution is a D-dimensional vector, and each component corresponds to a parameter that is to be optimized. Figure 1 illustrates the population and the vector candidates.

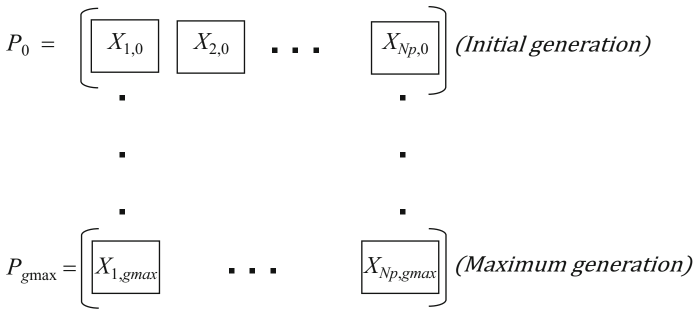


Fig. 1 Population and candidate structure

3.2.2 Initialization

DE starts the optimization process by generating an initial population P_o of N_p vectors encoded with D parameters, $\{X_{i,0} = (x_{j,i,0}), j = 0, \ldots, D - 1, i = 1, \ldots, N_p$. Every vector's parameter is initialized within the specified upper bound, ${x_j}^U$, and lower bound, ${x_j}^L$, of each parameter.

$$x_j^L < x_{j,i,g} < x_j^U \tag{19}$$

For example, the *j*th parameter of the *i*th vector may be initialized as follows:

$$x_{j,i,0} = \text{rand}_j\ (0, 1) \left(x_j^U - x_j^L\right) + x_j^L \tag{20}$$

where *rand*$_j$(0,1) is a uniformly distributed random number in the interval (0,1). It is also possible to initialize with non-uniform distributions, depending on what is known about the optimum location. If there is no prior information, a uniform distribution minimizes the chance of premature convergence and is preferred.

The initialized population is next subjected to mutation, as described in the following section.

3.2.3 Mutation

In biology, mutation is defined as a change in an organism's gene which results in an altered effect of its expression. This gene's alteration allows the organism to better adapt to its environment. In the context of DE, 'mutation' is defined as a process of taking a small random sample of vectors from the current population and combining them algebraically to form a new vector, which is referred to as a *mutant vector* [20]. In the so-called classical version of DE, the mutant vector is formed as follows:

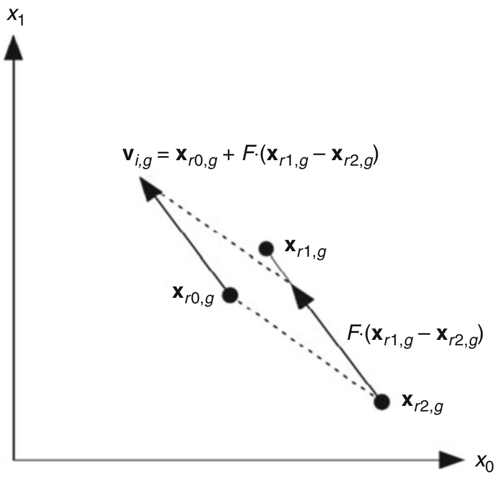

x_1
$\mathbf{v}_{i,g} = \mathbf{x}_{r0,g} + F\cdot(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g})$
$\mathbf{x}_{r1,g}$
$\mathbf{x}_{r0,g}$
$F\cdot(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g})$
$\mathbf{x}_{r2,g}$
x_0

Fig. 2 Differential mutation

$$V_{i,g} = X_{r0,g} + F\left(X_{r1,g} - X_{r2,g}\right), \tag{21}$$

where i, r_0, r_1, and r_2 are all distinct indices in the interval $[1,N_p]$. The mutation scale factor F is a positive real number between 0 and 2 that controls the rate at which the population evolves. The vector $X_{r0,\,g}$ is the *base vector*, while $X_{r1,\,g} - X_{r2,\,g}$ is the *difference vector*. The process of mutation is shown schematically in Fig. 2.

The above process is repeated N_p-times to constitute a mutant population.

In the original version (designated as **DE/rand/1**), each base vector is used only once per generation, in order to preserve diversity in the population. This version is widely used, although it has the drawback of relatively slow convergence [22, 51]. Some alternative mutation strategies to the classical version are as follows:

DE/best/1: This strategy resembles DE/rand/1, except that all mutants use the best vector in the current generation as the base vector:

$$V_{i,g} = X_{best,g} + F\left(X_{r1,g} - X_{r2,g}\right) \tag{22}$$

This approach has faster convergence than DE/rand/1, but often fails to reach the global optimum [51].

DE/best/2: This strategy uses two mutation differences to create a mutant vector:

$$V_{i,g} = X_{best,g} + F\left(X_{r1,g} - X_{r2,g}\right) + F\left(X_{r3,g} - X_{r4,g}\right) \tag{23}$$

where X_{r1}, X_{r2}, X_{r3}, X_{r4} are distinct random vectors and $X_{best,\,g}$ is the best individual of the current population. This approach attempts to balance between convergence speed and robustness. However, it still may converge to a local but non-global optimum, due to the fact that the base vector $X_{best,\,g}$ draws the population towards itself [22].

DE/local-to-best/2: This strategy resembles DE/best/2 in that two mutation differences are used: but the base vector is randomly sampled and the “best” vector is used in one of the scaled differences:

$$V_{i,g} = X_{ro,g} + F\cdot\left(X_{best,g} - X_{r1,g}\right) + F\cdot\left(X_{r2,g} - X_{r3,g}\right). \tag{24}$$

This approach has similar convergence properties to DE/best/2 [22].

DE/rand/2: This strategy samples 5 random vectors in the current generation to form two random differences which are scaled and added to the base vector:

$$V_{i,g} = X_{ro,g} + F\left(X_{r1,g} - X_{r2,g}\right) + F\left(X_{r3,g} - X_{r4,g}\right) \tag{25}$$

where $r_0 \neq r_1 \neq r_2 \neq r_3 \neq r_4$. This approach converges more slowly but is very robust [22, 51].

In this chapter, DE/rand/2 is used due to the aim set to appropriately tune the PSS with optimal time constants values for a robust performance.

Following mutation, vectors are next subject to crossover, as described in the next section.

3.2.4 Crossover

In biology, crossover is defined as the process of forming offspring by genetically combining two different parents [52]. In DE, ‘crossover’ refers to the process of creating a new vector (called the *trial vector*) by combining a mutant vector with a *target vector*. The target vector for mutant vector $V_{i,\,g}$ is $X_{i,\,g}$: note that according to the above description the mutant vector is constructed based on vectors $X_{r0,\,g}$, $X_{r1,\,g}$, ... which are all different from the target vector. The trial vector $U_{i,\,g} = [u_{1,\,i,\,g}, u_{2,\,i,\,g}, \ldots, u_{D,\,i,\,g}]$ is then obtained as follows:

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } \left(\text{rand}_j\,(0,1) \leq CR \text{ or } j = j_{rand}\right), j = 1, 2, \ldots, D \\ x_{j,i,g} & \text{otherwise} \end{cases} \tag{26}$$

where $CR \in [0, 1]$ is the *crossover probability*, CR is the fraction of the parameter values that are copied from the mutant vector, and $1 - CR$ is the fraction of parameter values copied from the trial vector. To determine whether the parameter to be copied is from the mutant or trial vector, a uniformly distributed random number $rand_j$ between [0,1] is generated and compared to the predefined value of CR. In addition, a random index $j_{rand} \in [1, N_p]$ is chosen and the corresponding mutant parameter copied to ensure that the trial vector is not a duplicate of the target vector.

The form of crossover described above is *uniform crossover*, and this is the form that was utilized in our implementation. There is an alternative crossover scheme called *exponential crossover*, which is described in Price et al. [22].

Once crossover is complete, selection is performed as described in the next section.

3.2.5 Selection

The selection process consists of choosing the individuals that will enter the next generation. In most EAs, some form of selection pressure is applied to ensure that the subsequent generation will consist of individuals that are generally more "fit" than the previous generation. DE employs a 'one-to-one survivor selection' which consists of comparing each trial vector to its corresponding target vector. Mathematically, the vector $X_{i,g+1}$ in the $g + 1$th generation is obtained from the trial vector $U_{i,g}$ and target vector $X_{i,g}$ as follows (in the case of a minimization problem: for a maximization problem the '$\leq$' is replaced with '$\geq$':

$$X_{i,g+1} = \begin{cases} U_{i,g} & \text{if } f\left(U_{i,g}\right) \leq f\left(X_{i,g}\right) \\ X_{i,g} & \text{otherwise} \end{cases} \tag{27}$$

This process ensures that the best vector at each index is retained. Furthermore, this also guarantees that the best-so-far solution is kept.

Once selection is performed for all target vectors in the current generation g, the processes of mutation, crossover, and selection are repeated with the N_p vectors in the $g +$ first generation. This process is iterated until a termination criterion is satisfied, as described in the next section.

3.2.6 Termination

There are several ways to terminate the DE optimization process. Some alternatives are listed below.

Objective Met

If the optimum value of the objective function is known, then the algorithm may be terminated when the best solution is within a specified tolerance of the optimum. In most practical cases, the optimum value is unknown, so this method cannot be applied.

Fixed Number of Generations

The optimization process is terminated after a fixed number of generations. Upon termination, the best known solution is reported. This was the method used in our implementation.

Population Statistics

The process is terminated when the difference between individuals with the worst fitness value and best fitness value is below a predetermined limit [22]. This is an indication that population is no longer diverse.

Limited Time

This is similar to 'fixed number of generations' above, except that execution time rather than number of generations is the limiting factor.

The performance of DE is affected by its control parameters: suboptimal parameter settings degrade the convergence of the algorithm. Optimal parameters may be obtained by trial and error, but this approach is costly in terms of computation time. Recently, adaptive or self-adaptive mechanisms have been introduced to dynamically and automatically update the parameters as the algorithm progress.

3.3 Self-Adaptive DE Algorithms

Self-adaptive algorithms have shown faster and more reliable convergence performance than the Classic DE (CDE) for many problems [26, 48, 53]. Some of these recently developed adaptive strategies are summarized below.

Fuzzy Adaptive Differential Evolution (FADE) was introduced in Lui and Lampinen [54]. It is a fuzzy logic-based algorithm which dynamically adapts the mutation and crossover operations, while the population is kept fixed.

Self-Adaptive Differential Evolution (SaDE) was proposed in Quin and Suganthan [55]. Two mutation strategies, 'DE/rand/1' (strategy 1) and 'DE/current-to-best/2' (strategy 2), may be used in this algorithm. The self-adaptive strategy consists of applying a probability p_i to first determine the mutation strategy (1 or 2) to be used to generate a mutation vector. This probability is updated after 50 generations according to the results obtained. The mutation and crossover parameters are also changed dynamically from generation to generation. Each mutation parameter is independently generated using a fixed normal distribution, while the crossover probabilities $CRi\ g$ for the ith individual in generation g is chosen as

$$CR_i = \text{rand}_n\,(CR_m, 0.1) \tag{28}$$

where CR_m is the mean of the successful *CR* values in the previous 25 generations.

jDE was introduced in Brest et al. [53]. As in the previous two schemes, jDE fixes the population size during the optimization while adapting the control parameters F_i and CR_i associated with each individual by means of an evolutionary scheme. The algorithm is based on DE/rand/1. Initially, $Fi = 0.5$ and $CRi = 0.9$ for each individual. jDE regenerates new values for *Fi* and *CRi* according to the following rules:

$$F_{i,g+1} = \begin{cases} F_l + \text{rand}_1\,[0,1] * F_u & \text{if rand}_2\,[0,1] < \tau_1 \\ F_{i,g} & \text{otherwise} \end{cases} \tag{29}$$

$$CR_{i,g+1} = \begin{cases} \text{rand}_3\,[0,1] & \text{if rand}_4\,[0,1] < \tau_2 \\ CR_{i,g} & \text{otherwise} \end{cases} \tag{30}$$

where

rand$_j$[0,1], $j = 1, 2, 3, 4$, are uniform random values on [0,1];
τ_1 and τ_2 represent the probabilities of adjusting the control parameters;
F_l and F_u are fixed lower and upper bounds, respectively, used for mutation.

In Brest et al. [53], the values of τ_1, τ_2, F_l, and F_u were chosen as 0.1, 0.1, 0.1, and 0.9, respectively.

Since the mutation and crossover parameters of successful individuals are carried over to the next generation, it is expected that ‘better’ values for *F* and *CR* will predominate in the population.

4 Population-Based Incremental Learning (PBIL)

4.1 Overview

Like most EAs, PBIL is an optimization technique that stochastically searches for the optimum value of a function by utilizing some aspects of GAs combined with competitive learning to guide its search towards prospective areas of the search space [9, 32–34, 44, 56]. Developed by Baluja in 1994 [32], PBIL uses bit strings to encode individual solutions, and uses probability vectors (PV) to guide the random generation of candidate bit strings and create other vectors through learning. PBIL then uses the current probability vector to create N_p individuals, which in turn are evaluated with an objective function. Using the best individuals of the current population, the probability vector is updated by shifting the likelihood of producing solutions corresponding to the best. There are several variant schemes for updating the probability vector: in this chapter we will present only one of the possibilities.

4.2 Binary Encoding, Probability Vector, and Population

PBIL requires that individual solutions be encoded as bit strings. If, for example, the number of parameters in a solution is n and the number of bits used to encode each parameter is l, then an individual will be encoded as a bit string of length m where $m{=}n \cdot l$. Probability vectors used in PBIL also have the same length:

$$PV = (p_1, p_2, \ldots, p_m)\,, \quad \text{where } m = nl,\ \ p_j \geq 0, \text{and} \sum\nolimits_{j=0}^{m} p_j = 1 \qquad (31)$$

For example, if a solution has 4 parameters encoded and each parameter is represented by 2 binary bits, then each probability vector PV will be constituted of 8 probability values, $PV = (p_1, p_2, \ldots, p_8)$.

Initially, each entry of the PV is set equal to 0.5, corresponding to equal probabilities of 0 and 1 at each position of the bit string. Using this PV, a first-generation population of N_p random bit strings of length m is generated, where each entry of each bit string is generated randomly, taking values 0 or 1 with equal probability.

4.3 Mutation

Instead of applying mutation to the population, PBIL applies mutation to the probability vector in order to prevent PV entries from converging too quickly to an extreme value (either 0.0 or 1.0). This helps prevent premature convergence to a local optimum [32].

There are many possible variants of the mutation operation. In this research the following procedure was followed. First, the mutation probability $p_{m\mathrm{ut}}$ *is* fixed. For each locus $j = \{1, \ldots, m\}$ of the PV, a random number $rand_j[0,1]$ is generated. If the generated value is less than the probability p_{mut}, then entry p_j is modified as follows:

$$p_j \leftarrow p_j + p_{mut}\left(0.5 - p_j\right). \qquad (32)$$

This adjustment shifts p_j towards the neutral value 0.5, and away from the extreme values 0 and 1. Using this mutated probability vector, a single bit string is randomly generated, such that the jth entry of the bit string takes the value 1 with probability p_j and the value 0 with probability $1 - p_j$. After this, another mutated PV is generated and used to generate another bit vector. This cycle is repeated until an entire new generation of N_p individuals is created.

4.4 Learning Process

In PBIL, the fitness of the population is made to increase from generation to generation by means of a learning process. The learning process involves a single parameter (denoted by α) called the *learning rate*, where $0 < \alpha < 1$.

The learning process is implemented within PBIL as follows. After a new population of N_p individuals is created, the fitnesses of all individuals are evaluated and the best individual is chosen. Recall that each individual is encoded as a bit string, so we may represent the best individual as a string $B_1 \dots B_m$ where each B_j is 0 or 1. This bit string is used to create a new PV $\left(p'_1, \dots, p'_m\right)$ as follows:

$$p'_j = (1 - \alpha)\, p_j + \alpha B_j, \quad j \in \{1, \dots, m\}. \tag{33}$$

A larger learning rate speeds up convergence, but it reduces the function space to be searched, while a smaller rate slows down the convergence but enables the exploration of a bigger search space, thereby increasing the likelihood of better optimal solutions [32, 44].

Note that PBIL has minimal memory requirements. Only the current PV and the current best solution for the present generation are stored. Therefore, it is a good candidate for online applications.

4.5 Termination

PBIL can use similar termination criteria as those described for DE above. In our implementation, we used a fixed number of generations.

5 Application of DE and PBIL to PSS Design

5.1 Overview

In this section, DE and PBIL are applied to obtain optimal PSS parameters based on the mathematical model described above. The optimization procedure involves the following steps for all system configurations.

Step 1. Define the range of operating conditions.

Step 2. Test them by running the load flow. Eliminate those conditions for which the load flow does not converge.

Step 3. Linearize the system at each operating condition and store the state-space matrices **A, B, C,** and **D** for each condition.

Step 4. Design the PSS by solving the constrained optimization problem given by Eqs. (16) and (17) using DE and PBIL. This is applied to all combined operating conditions.
Step 5. Once the optimum parameters are found, they are tested for robustness.
Step 6. In the case where the resulting PSS performance is not good, go to Step 1.

These steps are summarized by the flow charts in Figs. 3 and 4 representing the application of DE and PBIL, respectively.

Following the obtainment of PSS parameters based on DE and PBIL, their performances are evaluated by conducting a modal analysis. The results are then validated through time domain simulations. Furthermore, a comparison analysis is carried out for different designs and CPSS.

However, to obtain the optimum PSS parameters, both algorithms had to be configured. DE requires user-defined parameter inputs such as the mutation factor F, the crossover probability CR, the population size N_p, and the maximum number of generations g. These parameters greatly influence DE's ability to find the global minimum/maximum. F and CR values used were determined by trial and error. In our simulations $N_p = 50$ was found to give best results, which is half of the recommendation of [22] to use a population of 10 times the number of parameters to be optimized.

The user-defined inputs of PBIL are learning rate (α), forgetting factor (FF), as well as population size N_p and number of generations g as in DE. As in Sheetekela [44], α was set to be 0.1 and FF to be 0.005. N_p was set to 50 as for DE, and g was set to 300 or 500 (for the two different systems modelled) to allow for time to convergence [7, 8, 44, 56].

Table 1 summarizes the parameters used for both algorithms.

5.2 System Configurations

In this chapter, two electrical power system configurations were considered for the PSS design: the single machine to infinite bus (SMIB) and the two-area multimachine (TAM). Brief descriptions of the two systems are given below. For complete specifications of the two configurations, and for the mathematical formulations of the corresponding PSS optimization problems, see [57].

The SMIB system consists of a synchronous generator connected to the infinite bus through a double transmission line, as shown in Fig. 5. The generator was modelled using a system of six differential equations, and is equipped with a simple exciter and turbine governor which were both modelled using differential equations. The operating conditions of this system have been obtained by simulating variations of generator output and transmission line reactance.

The TAM system consists of two areas, each with identical generating units and similar ratings as shown in shown in Fig. 6. As in the SMIB system, the generators were equipped with simple exciters and turbine governors, and were modelled using

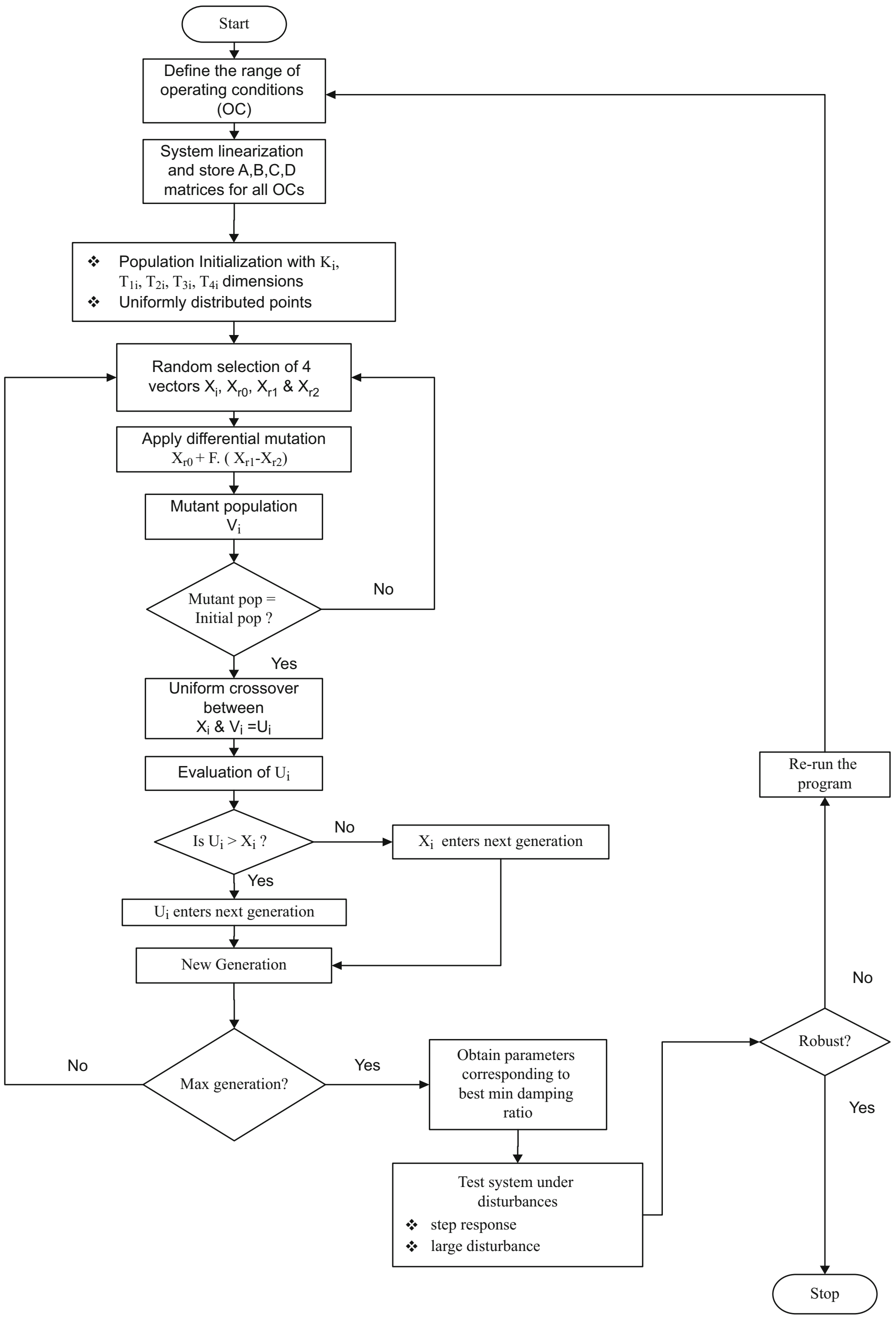

Start
Define the range of operating conditions (OC)
System linearization and store A,B,C,D matrices for all OCs
❖ Population Initialization with K_i, T_{1i}, T_{2i}, T_{3i}, T_{4i} dimensions
❖ Uniformly distributed points
Random selection of 4 vectors X_i, X_{r0}, X_{r1} & X_{r2}
Apply differential mutation $X_{r0} + F. (X_{r1}-X_{r2})$
Mutant population V_i
Mutant pop = Initial pop ?
No
Yes
Uniform crossover between X_i & $V_i = U_i$
Evaluation of U_i
Is $U_i > X_i$?
No
X_i enters next generation
Yes
U_i enters next generation
New Generation
Max generation?
No
Yes
Obtain parameters corresponding to best min damping ratio
Test system under disturbances
❖ step response
❖ large disturbance
Robust?
No
Re-run the program
Yes
Stop

Fig. 3 Flowchart for the PSS design using DE

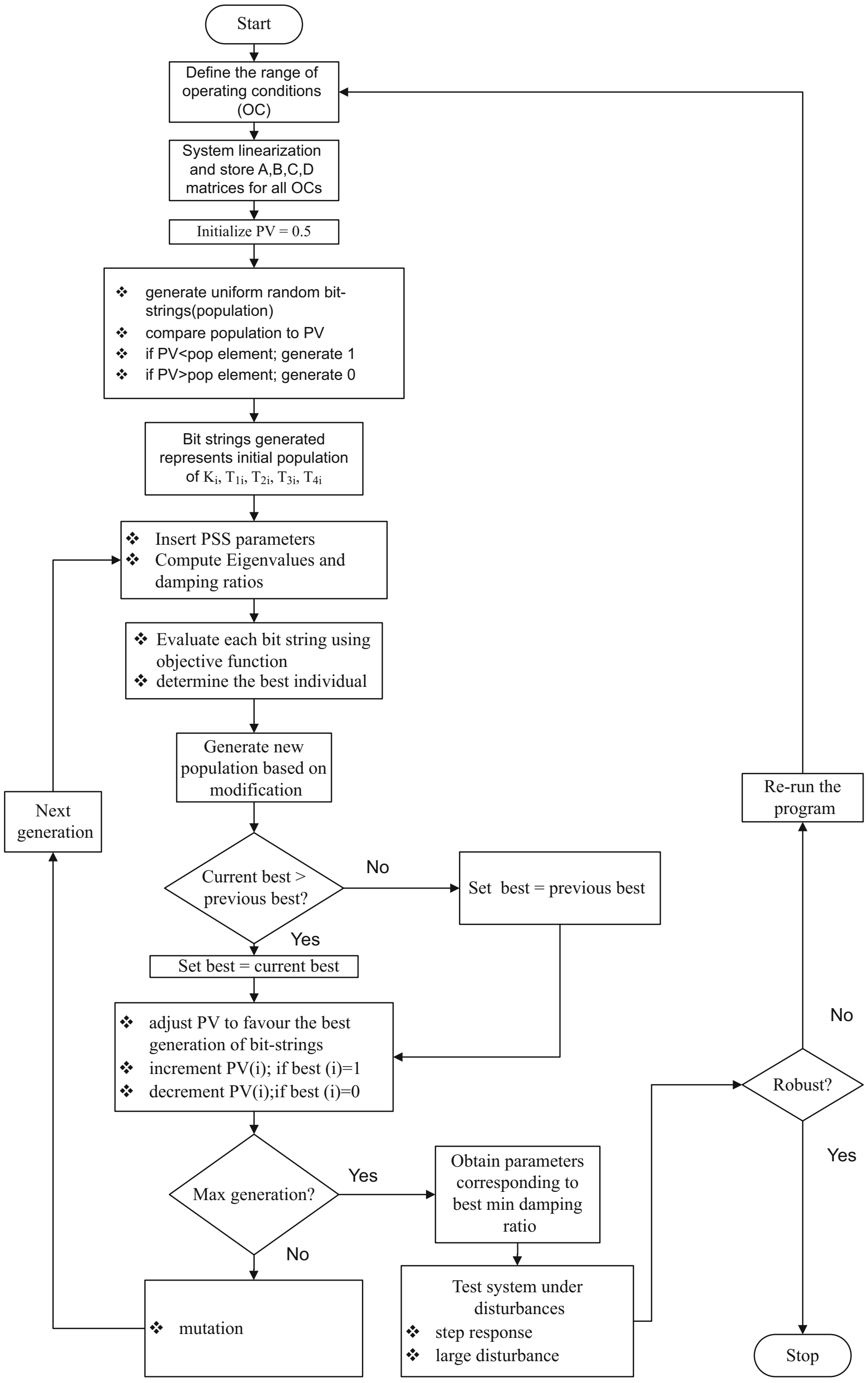

Start
Define the range of operating conditions (OC)
System linearization and store A,B,C,D matrices for all OCs
Initialize PV = 0.5
❖ generate uniform random bit-strings(population)
❖ compare population to PV
❖ if PV<pop element; generate 1
❖ if PV>pop element; generate 0
Bit strings generated represents initial population of K_i, T_{1i}, T_{2i}, T_{3i}, T_{4i}
❖ Insert PSS parameters
❖ Compute Eigenvalues and damping ratios
❖ Evaluate each bit string using objective function
❖ determine the best individual
Generate new population based on modification
Current best > previous best?
No
Set best = previous best
Yes
Set best = current best
❖ adjust PV to favour the best generation of bit-strings
❖ increment PV(i); if best (i)=1
❖ decrement PV(i);if best (i)=0
Max generation?
Yes
Obtain parameters corresponding to best min damping ratio
No
❖ mutation
Next generation
Test system under disturbances
❖ step response
❖ large disturbance
Robust?
No
Re-run the program
Yes
Stop

Fig. 4 Flowchart for the PSS design using PBIL

Table 1 DE and PBIL parameters

Parameter	Value used in DE	Value used in PBIL
Population	50	50
# of generations	200	300 (SMIB) or 500 (TAM)
Termination	Fixed # of generations	Fixed # of generations
Mutation *F*	0.95	–
Crossover *CR*	0.95	–
Length of encoded solution	–	15 bits
Learning rate α	–	0.1
Forgetting factor *FF*	–	0.005

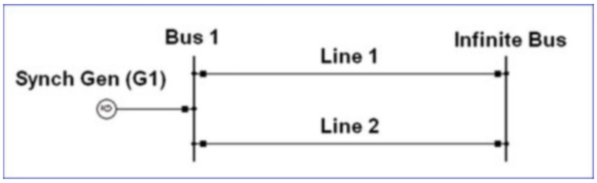


Fig. 5 Single machine to infinite bus (SMIB) system diagram

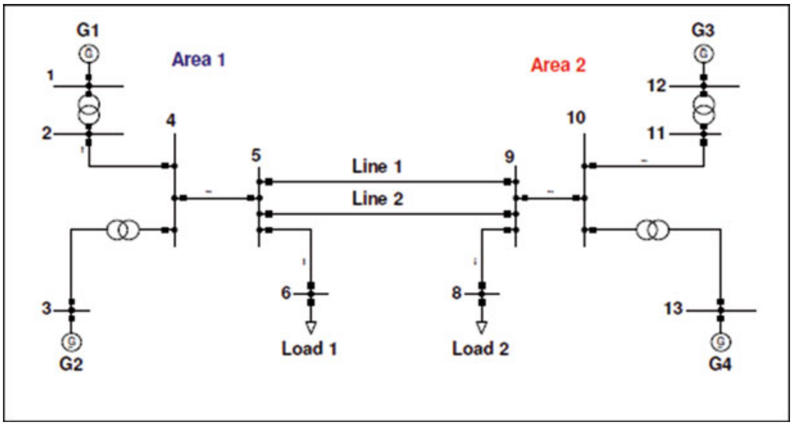


Fig. 6 Two-area multimachine (TAM) system line diagram. G1-G4 are the generators. The linked circle symbols indicate transformers

the same differential equations. The system has two loads connected to buses 6 and 8, as shown in the figure. The two areas are connected by two tie-lines. Six different operating conditions were specified based on varying the load demand, which subsequently varied the power transferred over the tie-lines.

5.3 Single Machine Infinite Bus System: Results of Optimization

Figure 7 shows the fitness (minimum damping ratio) for the SMIB system as a function of generation number when DE optimization is applied. The DE algorithm converged to an optimum minimum damping ratio value of 0.2659. DE reached its final value after only 100 generations of its complete run of 200 generations.

On the other hand, Fig. 8 shows the corresponding fitness curve for PBIL applied to the same system. PBIL converged to an optimum damping ratio value of 0.2325. PBIL starts settling after around 150 generations, over a maximum algorithm duration of 300 generations. This difference reflects the slower convergence of PBIL as a function of generation number.

Table 2 shows the PSS parameters for the SMIB system obtained via DE, PBIL, and CPSS methods.

Table 3 shows the eigenvalues corresponding to the electromechanical modes for the five different operating conditions used, and the respective damping ratio in brackets. On average, the DE damping ratios were larger (hence better) than corresponding PBIL and CPSS values by an average of about 18% and 51%, respectively.

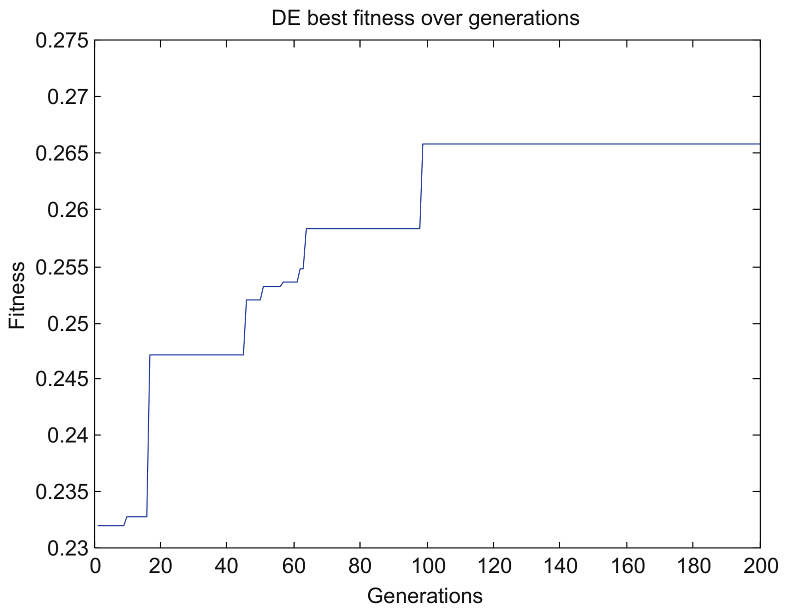


Fig. 7 DE fitness curve for PSS optimization, single machine infinite bus system

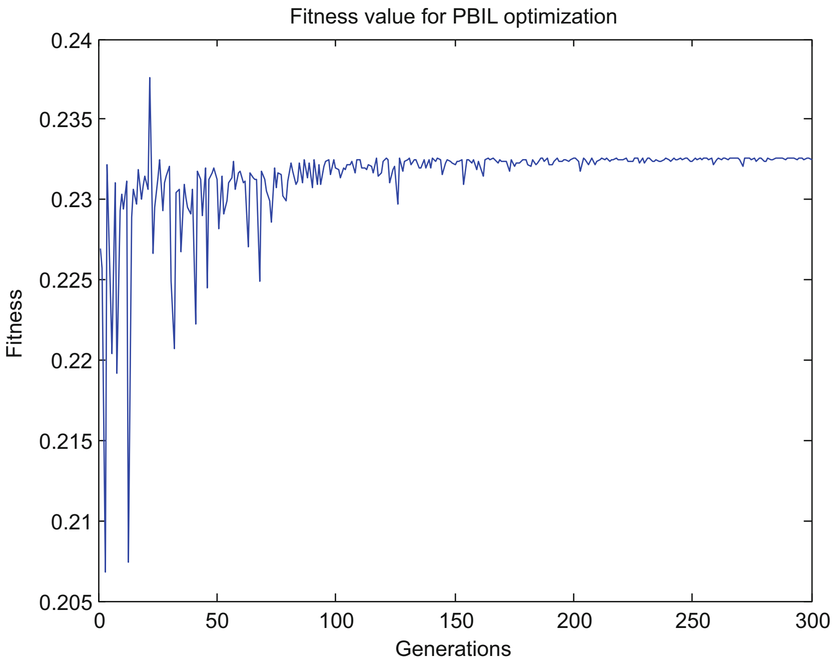


Fig. 8 PBIL fitness curve for PSS optimization, single machine infinite bus system

Table 2 SMIB PSS parameters

Algorithm	K_p	T_1	T_2	T_3	T_4
CPSS	13.868	3.7440	0.8778	3.7440	0.8778
DE-PSS	19.985	4.9376	0.4339	0.0103	0.1290
PBIL	16.361	1.7217	1.8110	4.9435	0.5568

5.4 Two-Area Multimachine System: Results of Optimization

Figure 9 shows the fitness (minimum damping ratio) for the TAM system as a function of generation number when DE optimization is applied. The DE algorithm attains its optimum minimum damping ratio value of 0.2263 after 140 generations, out of a complete run of 180 generations.

Figure 10 shows the PBIL fitness curve for the same system. PBIL converged to an optimum damping ratio value of 0.2094. In this case, PBIL continues to show improvements for about 400 generations, which is somewhat longer than for the single machine infinite bus system.

Table 4 shows the DE and PBIL-optimized PSS parameters for the TAM system, along with the CPSS parameters.

Table 5 shows the eigenvalues corresponding to the inter-area mode for various system operating conditions. The respective damping ratios are in parentheses. The DE-PSS values are about 10% and 52% higher than the corresponding PBIL-PSS

Table 3 SMIB electromechanical modes of the system with different PSS

Cases	DE-PSS	PBIL-PSS	CPSS	No PSS
1	$-1.6950 \pm j2.7780$ (0.5210)	$-1.3296 \pm j2.6279$ (0.4515)	$-1.1033 \pm j2.9792$ (0.3473)	$-0.1770 \pm j3.7640$ (0.0470)
2	$-1.2323 \pm j2.4201$ (0.4537)	$-0.9570 \pm j2.2621$ (0.3896)	$-0.8089 \pm j2.5271$ (0.3049)	$-0.1512 \pm j3.3950$ (0.0445)
3	$-1.1576 \pm j2.7233$ (0.3917)	$-0.8936 \pm j2.8265$ (0.3014)	$-0.7281 \pm j3.0379$ (0.2331)	$-0.1190 \pm j3.2450$ (0.0370)
4	$-0.9102 \pm j2.2167$ (0.3798)	$-0.7160 \pm j2.0048$ (0.3363)	$-0.5957 \pm j2.2107$ (0.2602)	$-0.1080 \pm j2.9400$ (0.0360)
5	$-0.7065 \pm j2.5401$ (0.2659)	$-0.6520 \pm j2.6040$ (0.2325)	$-0.5114 \pm j2.7087$ (0.1855)	$-0.0760 \pm j2.7810$ (0.0270)

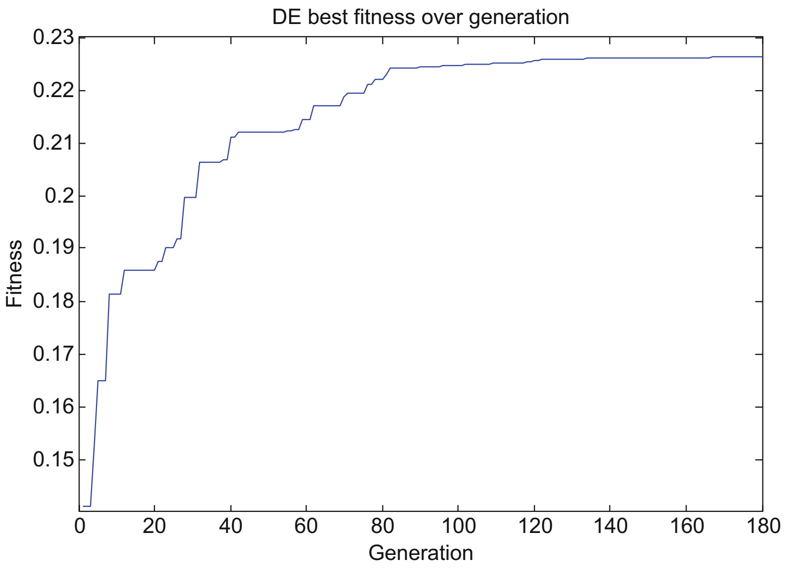


Fig. 9 DE fitness curve for PSS optimization, two-area multimachine system

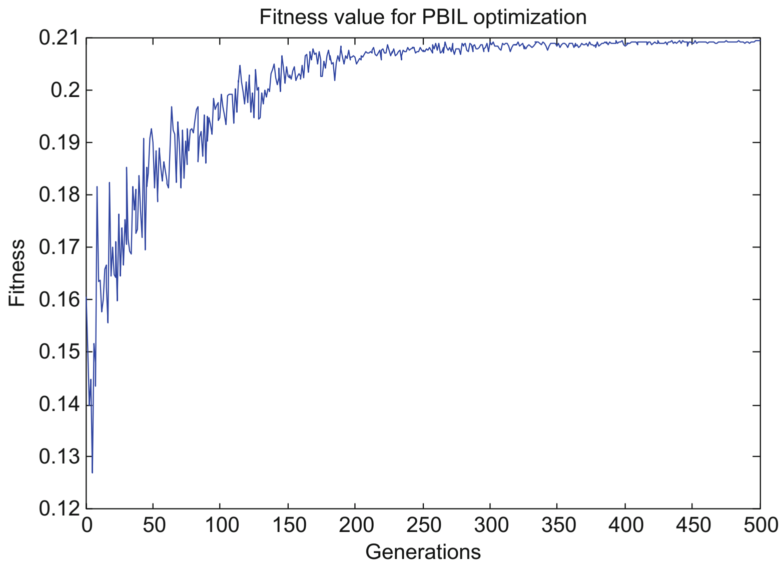


Fig. 10 PBIL fitness curve for PSS optimization, two-area multimachine system

and CPSS values, respectively. The system also has two local area modes, but these had consistently higher damping ratios than the inter-area mode, so it was the inter-

Table 4 PSS parameters for TAM system

		K_p	T_1	T_2	T_3	T_4
PBIL-PSS	*Gen 1&2*	19.191	0.238	0.012	0.049	0.014
	Gen 3&4	16.633	0.119	0.083	0.055	0.010
DE-PSS	*Gen 1&2*	19.992	0.051	0.019	0.053	0.015
	Gen 3&4	19.997	0.116	0.015	0.111	0.016
CPSS	*Gen 1&2*	9.877	0.307	0.015	0.051	0.013
	Gen 3&4	13.685	0.126	0.085	0.062	0.010

area mode that determined system stability. For exact figures and additional details, see [57].

5.5 Sensitivity of Differential Evolution to Algorithm Control Parameters.

In the previous section, we saw that DE optimization gave the best PSS parameters, when compared to PBIL and CPSS. We mentioned previously that the DE optimization process is heavily dependent on the control parameters: mutation factor, crossover probability, and population size. In this section we investigate the effects of these three parameters on the algorithm's performance by varying the parameters independently.

5.5.1 Effects of *F* and *CR* Parameters on DE Convergence

Figure 11 shows the effect of varying F (with *CR* fixed at 0.9) on the optimum value obtained for the two-area multimachine system. Results show that *F* in the range of 0.75–0.9 gives the best performance. In Fig. 12, *F* is kept constant at 0.9 and *CR* is varied. Results show that $F = 0.9$ and $CR = 0.95$ produce the optimal performance.

5.5.2 Effect of Population Size

Population size has a huge influence on the convergence properties of the DE algorithm.

If the population size is small, the algorithm may converge faster, but the probability of premature convergence is high. Smaller populations are also more prone to a phenomenon known as *stagnation,* where the population remains diverse and unconverged, while the optimization process no longer progresses. Larger populations are less susceptible to premature convergence or stagnation, but require more computational effort.

Table 5 Inter-area modes for TAM system

Case	DE-PSS	PBIL-PSS	CPSS	No-PSS
1	$-1.273 \pm j4.528$ (0.2707)	$-1.115 \pm j4.589$ (0.2362)	$-0.807 \pm j4.509$ (0.1761)	$-0.0060 \pm j4.962$ (0.0012)
2	$-1.219 \pm j4.507$ (0.2612)	$-1.062 \pm j4.566$ (0.2265)	$-0.753 \pm j4.484$ (0.1656)	$0.0461 \pm j4.937$ (-0.0093)
3	$-1.154 \pm j4.477$ (0.2492)	$-1.044 \pm j4.534$ (0.2245)	$-0.741 \pm j4.453$ (0.1642)	$0.0428 \pm j4.894$ (-0.0087)
4	$-1.089 \pm j4.436$ (0.2384)	$-1.022 \pm j4.491$ (0.2218)	$-0.725 \pm j4.412$ (0.1623)	$0.0806 \pm j4.792$ (-0.0165)
5	$-1.015 \pm j4.404$ (0.2263)	$-0.950 \pm j4.420$ (0.2098)	$-0.648 \pm j4.384$ (0.1506)	$0.0771 \pm j4.697$ (-0.0168)
6	$-0.806 \pm j3.477$ (0.2259)	$-0.748 \pm j3.496$ (0.2094)	$-0.518 \pm j4.411$ (0.1502)	$0.0589 \pm j3.419$ (-0.0170)

Damping ratios are in parentheses

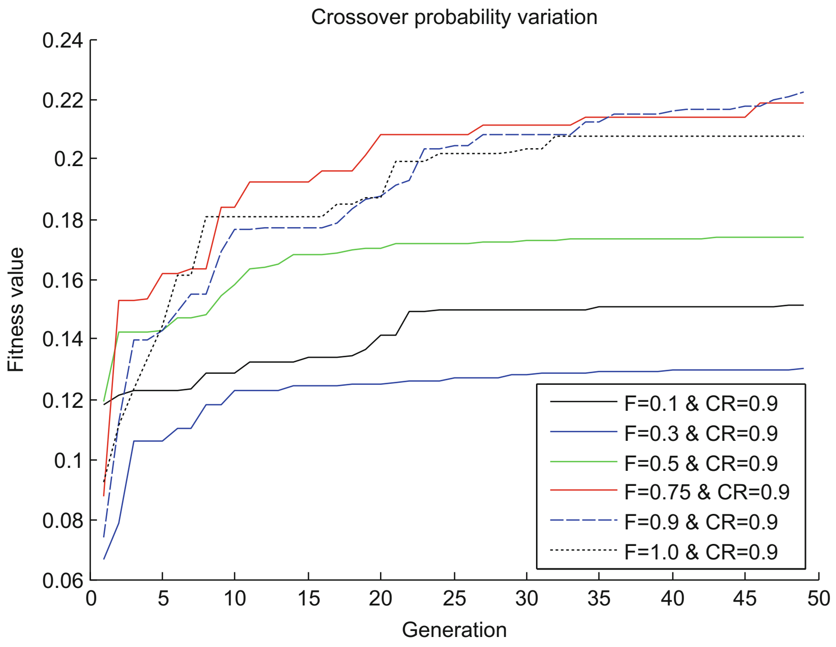


Fig. 11 Effect of the mutation parameter F on DE performance (CR = 0.9)

Table 6 Experimental results of DE when varying the population

Population size	Best damping	Mean	Std. dev.
10	0.1780	0.1610	0.0163
30	0.2439	0.2268	0.0224
50	0.2694	0.2301	0.0110
70	0.2599	0.2214	0.0235
100	0.2671	0.2254	0.0322
150	0.2740	0.2458	0.0282
200	0.2603	0.2231	0.0267

The authors in Price et al. [22] recommended that a population between *7.D* and *10.D* be used for a good and efficient performance, where D is the number of parameters to be optimized. The influence of population size grows larger with increasing number of parameters [22, 58–61].

The investigations in this research are carried out on a TAM system with $D = 10$ parameters. Different population sizes from $3D$ to $20D$ were used, and each population size was run 10 times independently for 200 generations. Experimental results recorded in Table 6, show the best and mean damping as well as the standard deviations obtained from the 10 runs.

Table 6 shows that a population of 150 (i.e., 15D) gave the highest best and mean values for the damping ratio. This case also gave lower standard deviation than the results for the population size of 100 ($10D$).

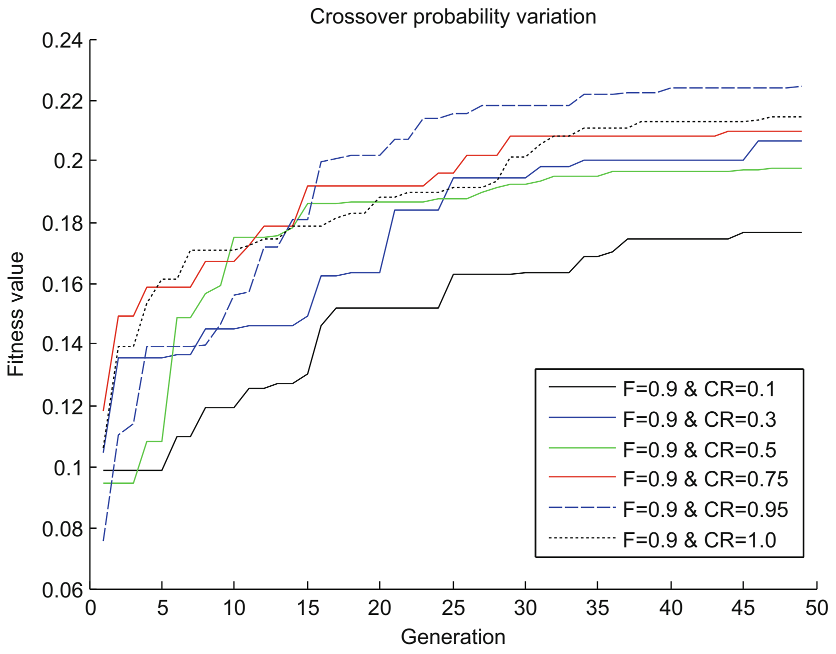


Fig. 12 Effect of CR probability on DE performance ($F = 0.9$)

5.6 Application of Adaptive DE to PSS Design

We employed an adaptive scheme similar to jDE, but based on the mutation strategy DE/rand/2 for improved robustness. Specifically, the mutation process is carried out according to:

$$V_{i,g} = X_{ro,g} + F_{i,g}^{1}\left(X_{r1,g} - X_{r2,g}\right) + F_{i,g}^{2}\left(X_{r3,g} - X_{r4,g}\right) \tag{34}$$

where $F_{i,g}^{1}$ and $F_{i,g}^{2}$ are generated independently based on Eq. (29). There are also two corresponding crossover parameters $CR_{i,g}^{1}$ and $CR_{i,g}^{2}$, which are generated according to Eq. (30). The values used for τ_1, τ_2, F_l, and F_u were chosen as 0.1, 0.1, 0.1, and 0.9, respectively, as in Brest et al. [53].

This adaptive algorithm was applied to the SMIB system, and the resulting PSS was compared to the PSS obtained using classical DE (CDE) based on the DE/rand/1 mutation strategy. The DE control parameters were configured as described in Table 7.

The PSS were designed over four operating conditions as shown in Table 8. This table also shows the PSS parameters obtained after the optimization using jDE and the CDE, and Table 9 shows the eigenvalues and damping ratios obtained when the

Table 7 Control parameters for DE and jDE algorithms

Parameter description	DE parameter values	jDE parameter values
Population	30	30
Generation	100	100
Mutation	0.9	Adaptive
Crossover	0.9	Adaptive
Termination	Max gen	Max gen

Table 8 PSS parameters

	K_p	T_1	T_2	T_3	T_4
CDE-PSS	17.2	0.0102	0.154	4.839	0.272
jDE-PSS	18.9	4.64	1.67	3.21	1.50

resulting PSSs are used for the different operating conditions. The table shows that jDE consistently achieves higher damping ratios than CDE.

5.7 Performance Summary

The performance of the DE optimization algorithm on the PSS tuning problem is strongly dependent on the settings of intrinsic algorithm parameters: mutation factor, crossover probability, and population size. Low values of F cause DE to prematurely converge to local optimum, whereas values between [0.9, 0.95] ensure good performance of the algorithm.

The crossover probability (CR) is responsible for the diversity within a given population. It is observed that low values of CR prevent DE from exploring the entire search space, possibly leading to stagnation. However, when CR is set between [0.9–0.95], the algorithm is able to both maintain population diversity and escape stagnation, which subsequently search better for the global optimum. CR should not be set to 1 because it may lead to a rapid loss of diversity.

The population size also influences the diversity. Small population sizes tend to lead to premature convergence, whereas larger populations tend to give slower convergence rates but better final results.

The jDE whereby F and CR are changing and Np is fixed was implemented to tune the PSS in SMIB. The results are compared with those of the CDE. The modal analysis shows that PSS designed with self-adaptive DE outperformed those from CDE for all the cases considered. These results have been validated with time domain simulations under small disturbances. However, these results are not shown here.

Table 9 System open-loop and closed-loop eigenvalues (damping ratios are in parentheses)

Case	Active Power (pu)	Line reactance (pu)	No PSS	CDE-PSS	jDE-PSS
Case 1	1.0	0.3	−0.2680 ± *j*4.457 (0.0600)	−1.521 ± *j*3.405 (0.408)	−1.920 ± *j*3.979 (0.434)
Case 2	1.0	0.5	−0.1180 ± *j*3.781 (0.0483)	−1.134 ± *j*2.735 (0.383)	−1.566 ± *j*3.232 (0.436)
Case 3	1.0	0.7	−0.1330 ± *j*3.311 (0.0402)	−0.832 ± *j*2.319 (0.330)	−1.341 ± *j*2.689 (0.446)
Case 4	1.0	0.9	−0.0997 ± *j*2.9470 (0.0338)	−0.487 ± *j*1.694 (0.276)	−1.161 ± *j*2.249 (0.458)

6 Chapter Summary

The problem of optimally tuning the power system stabilizer has been addressed in this chapter. Optimization techniques based on computational intelligence have been used to find a set of PSS parameters that provides superior damping for a wide range of operating conditions. In the first part of the chapter, DE was developed and implemented to tune the PSS. The algorithm performances were evaluated by comparing it to PBIL and CPSS. In the second part of the chapter, empirical rules were formulated with regard to DE intrinsic parameters that ensured optimal performance of the algorithm. To substitute the trial-and-error tuning approach, an adaptive scheme was applied to the DE control parameters and compared to classic DE.

References

1. P. Kundur, *Power System Stability and Control* (Prentice-Hall, s.l., 1994)
2. P.M. Anderson, A.A. Fouad, *Power System Control and Stability* (The Iowa State University Press, Ames, IA, 1994)
3. P. Bikash, C. Balarko, *Robust Control in Power Systems* (Springer Science, s.l., 2005)
4. G. Rogers, *Power System Oscillations* (Kluwer Academic, Boston, 1999)
5. N. Mithulananthan et al., Comparison of PSS, SVC, and STATCOM controllers for damping power system oscillations. IEEE Trans. Power Syst. **18**, 786–792 (2003)
6. G. Rogers, Demystifying power system oscillations. IEEE Comput. Appl. Power **9**, 30–35 (1996)
7. K.A. Folly, Multimachine power system stabilizer design based on a simplified version of genetic algorithm combined with learning, in *International Conference on Intelligent Systems Application to Power Systems (ISAP)* (2005)
8. K.A. Folly and S. Sheetekela, Application of Simple Estimation of Distribution Algorithm to Power System Controller Design, presented at the 45th International Universities Power Engineering Conference (UPEC) (2010)
9. KA. Folly, Small-signal stability enhancement of power systems using PBIL based power system stabilizer, in *Artificial Intelligence in Energy Systems and Power* (2006)
10. P. Kundur et al., Application of power system stabilizers for enhancement of overall system stability. IEEE Trans. Power Syst. **4**, 614–626 (1989)
11. Y.N. YU, K. Vongsuriy, L.N. Wedman, Application of optimal theory to power system. IEEE Trans. Power Apparatus Syst. **89**, 55–62 (1970)
12. T.C. Yang, Applying optimisation method to power system stabiliser design part 1: Single-machine infinite-bus systems. Electr. Power Energy Syst. **19**, 29–35 (1997)
13. E.V. Larsen, D.A. Swann, Applying power system stabilizers, part I; general concepts, part II; performance objectives and tuning concepts, part III; practical considerations. IEEE Trans. Power Apparatus Syst. **PAS-100**, 3017–3046 (1981)
14. J. Reddy, J.K. Mendiratta, H-infinity loop shaping based robust power system stabilizer for dynamic equivalent multi-machine power system, in *IEEE International Energy Conference* (2010)
15. S. Chen, O.P. Malik, H-infinity optimisation-based power system stabiliser. IEE Proc. Gener. Transm. Distrib. **142**, 179–184 (1995)
16. K.A. Folly, N. Yorino, H. Sasaki, Improving the robustness of H-infinity PSSs using the polynomial approach. IEEE Trans. Power Syst. **13**, 1359–1364 (1998)

17. J. Nocedal, S.J. Wright, *Numerical Optimization* (Springer-Verlag, New York, 1999)
18. S.Y. Li, S.S. Lee, Y.T. Yoon, J.K. Park, Non-linear adaptive decentralised stabilization control for multimachine power systems. Int. J. Control Autom. Syst. **7**, 389–397 (2009)
19. K.A. Folly, On the prevention of pole-zero cancellations in H infinity power system controller design: A comparison. SAIEE Afr. Res. J. **99** (2008)
20. K.A. Folly, Robust controller design for small-signal stability enhancement of power systems, in *IEEE AFRICON* (2004), pp. 631–636
21. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, Reading, MA, 1989)
22. K. Price, R. Storn, J. Lampinem, *Differential Evolution: A Practical Approach to Global Optimization* (Springer-Verlag, Berlin, 2005)
23. E. Mezura-Montes, J. Velazquez-Reyes C.A. Coello, A comparative study of differential evolution variants for global optimization, in *GECCO* (s.n., Seattle, WA, 2006), pp. 485–492
24. J. Vesterstroem, R. Thomsen, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, in *IEEE Congress on Evolutionary Computation* (2004), pp. 1980–1987
25. A.E. Eiben, J.E. Smith, *Introduction to Evolutionary Computing* (Springer, s.l., 2003)
26. S.M. Islam, An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. IEEE Trans. Syst. Man Cubernetics **42**, 482–500 (2012)
27. J. Brest, B. Boskovic, V. Zumer, An improved self-adaptive differential evolution algorithm in single objective constrained real-parameter optimization, in *Congress on Evolutionary Computation* (2010), pp. 1–8
28. J. Montgomery, Crossover and the different faces of differential evolution searches, in *IEEE Congress on Evolutionary Computation (CEC)* (2010)
29. J. Lampinen, I. Zelinka, On stagnation of differential evolution algorithm, in *Proc. of MENDEL, 6th Int. Mendel Conf. on Soft Computing*, ed. by O. Pavel, (s.n., Brno, 2000)
30. R. Storn, On the usage of differential evolution for function optimization, in *Fuzzy Information Processing Society* (1996), pp. 519–523
31. S. Rahnamayan et al., Opposition-based differential evolution. IEEE Trans. Evol. Comput. **12**, 64–79 (2008)
32. S. Baluja, Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report CMU-CS-94-163 (Carnigie Mellon University, 1994)
33. S. Yang, X. Yao, Population-based incremental learning with associative for dynamic environments. IEEE Trans. Evol. Comput. **12**, 542–561 (2008)
34. S.L. Ho, S. Yang, A population-based Incremental learning method for robust optimal solution. IEEE Trans. Magn. **46**, 3189–3192 (2010)
35. S.Y. Yang et al., A new implementation of population based incremental learning method for optimizations in electromagnetics. IEEE Trans. Magn. **43**, 1601–1604 (2007)
36. S. Baluja, R. Curuana, Removing the genetic from the standard genetic algorithm, in *Proceedings of the 12th International Conference on Machine Learning* (s.n., Lake Tahoe, 1995)
37. K.A. Folly, Robust controller design based on a combination of genetic algorithms and competitve learning, in *Proceedings of International Joint Conference on Neural Networks* (s.n., Orlando, FL, 2007)
38. Y.L. Abdel-Magid, A. Abido, H. Mantaurym, Simultaneous stabilization of multimachine power system via genetic algorithm. IEEE Trans. Power Sys **14**, 1428–1438 (1999)
39. Z. Michalewicz, *Genetic Algorithms+Data Structure=Evolution Programs*, 3rd edn. (Springer-Verlag, s.l., 1996)
40. P. Mitra et al., Comparative of population based techniques for power system stabilizer design, in *15th International Conference on Intelligent System Applications to Power Systems ISAP'09* (2009)

41. A. Qing, *Differential Evolution: Fundamental and Applications in Electrical Engineering* (John Wiley & Sons, s.l., 2009)
42. Z. Wang et al., Robust power system stabilizer design under multi-operating conditions using differential evolution. IET Gener. Transm. Distrib. **2**, 690–700 (2008)
43. T. Mulumba, K.A. Folly, Design and comparison of multi-machine power system stabilizer base on evolution algorithms, in *Universities' Power Engineering Conference (UPEC), Proceedings of 2011 46th International* (s.n., Soest, 2011), pp. 1–6
44. S.P.N. Sheetekela, Design of Power System Stabilizers Using Evolutionary Algorithms, Thesis (Electrical, University of Cape Town, Cape Town, 2010)
45. H. Shayeghi, A. Safari, H.A. Shayanfa, Multimachine power system stabilizers design using PSO algorithm. Int. J. Electr. Electron. Eng. **4**, 226–233 (2009)
46. M.A. Abido, Robust design of multimachine power system stabilizers using simulated annealing. IEEE Trans. Energy Convers. **15**, 297–304 (2000)
47. R. Storn, K. Price, Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**, 341–359 (1997)
48. S. Zhang, F.L. Luo, An improved simple adaptive control applied to power system stabilizer. IEEE Trans. Power Electron. **24**, 369–375 (2009a)
49. J. Zhang, A.C. Sanderson, *Adaptive Differential Evolution: A Robust Approach to Multimodal Problem Optimization* (Springer, Berlin, 2009)
50. R. Storn, K. Price, *Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Space* (International Computer Science Institute, Berkeley, 1995)
51. Y. Ao, H. Chi, Experimental study on differential evolution strategies, in *Global Congress on Intelligent Systems* (2009)
52. Science daily. Chromosomal crossover. Science Daily (8 June 2012). [Online] http://www.sciencedaily.com/articles/c/chromosomal_crossover.htm
53. J. Brest et al., Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Trans. Evol. Comput. **10**, 646–657 (2006)
54. J. Lui, J. Lampinen, A fuzzy adaptive differential evolution algorithm, in *Soft Computation: Fusion Found Method Appl.* (s.n., Brno, 2005), pp. 448–462
55. A.K. Quin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in *Congress on Evolutionary Computation* (2005), pp. 1785–1791
56. K.A. Folly, G.K. Venayagamoorthy, Optimal tuning of system stabilizer parameters using PBIL with adaptive learning rate, in *Power and Energy Society General Meeting* (IEEE, s.l., 2010), pp. 1–6
57. T. Mulumba, *Application of Differential Evolution to Power System Stabilizer Design, MSc. Dissertation* (University of Cape Town, 2012)
58. U.K. Chakraborty, *Advances in Differential Evolution* (Springer-Verlag, s.l., 2008)
59. R. Mallipeddi, P.N. Suganthan, Empirical study on the effect of population size on differential, in *IEEE Congress on Evolutionary Computation* (2008), pp. 3663–3670
60. J. Teo, *Exploring Dynamic Self-Adaptive Populations in Differential Evolution* (Springer-Verlag, s.l., Soft Comput, 2005), pp. 673–686
61. J. Brest, M.S. Maucec, Population size reduction for the differential evolution algorithm. Appl. Intell. **29**, 228–247 (2008)

Automatic Sign Language Manual Parameter Recognition (I): Survey

Mehrdad Ghaziasgar, Antoine Bagula, Christopher Thron, and Olasupo Ajayi

1 Background and Motivation

The South African Constitution recognises South African Sign Language (SASL) as the official language of the South African Deaf [1]. It has been estimated that as many as 235,000 people in South Africa are profoundly deaf in both ears and use SASL as their first and only language [2]. While this is the case, public infrastructure and resources in South Africa still remain largely out of reach for the South African Deaf. Very few resources, if any, have been adapted to the needs of this group of people. It may not be immediately apparent that the South African Deaf are mostly unable to communicate in spoken languages of any form, attributed to a number of realities that are mostly unknown to the hearing due to limited contact with the Deaf community.

The first sign language transcription system was published by William Stokoe in 1960 [3]. His system uses Latin alpha-numeric characters to symbolise gesture sequences. Other notations were later invented, including the Hamburg Notation System [4], which is an extension of Stokoe, and Sutton SignWriting (or SignWriting for short) which is a pictographic sign language transcription notation [5].

The translation of any sign language into English involves the following processes: capturing sign language gesture input, either visually or in other forms, recognising the gestures within the input data in order to arrive at a transcription of the gestures in a sign language computer-readable textual format, performing a

M. Ghaziasgar (✉) · A. Bagula · O. Ajayi
Department of Computer Science, University of the Western Cape, Cape Town, South Africa
e-mail: mghaziasgar@uwc.ac.za

C. Thron
Department of Science and Mathematics, Texas A&M University-Central Texas, Killeen, TX, USA

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_3

linguistic translation of the sign language transcription into English text, and finally using speech synthesis techniques to render English audio.

Referring to the capturing of sign language gestures, two main approaches have been used in the literature: auxiliary approaches and contactless approaches [6–8]. Auxiliary approaches make use of specialised wearable hardware such as data gloves, data suits, and/or coloured markers to simplify input capture. This makes it possible to obtain very accurate input such as the location and rotation of each joint in the hands. This comes at the expense of hardware cost, simplicity, user convenience, and naturalness of use [7]. On the other hand, contactless approaches do not require the user to wear any specialised equipment, preferring to use one or more cameras to capture video input. In particular, monocular-view approaches capture input from only a single monocular camera. This provides a simple, low-cost, and natural interface to users, but places exceptional demands on software. For this reason, gesture recognition from a monocular view is still a largely unsolved problem [6, 9].

This paper is concerned specifically with processing of SASL video captured from a monocular view to produce transcriptions of the SASL gestures in a computer-readable sign language transcription notation. This requires for an analysis of SASL input video to accurately extract SASL semantic information.

Stokoe showed that five parameters fully characterise any sign language gesture [3]. They are: hand location, hand motion, hand shape, hand orientation, and facial expression. The five parameters pertaining to the hands are collectively referred to as sign language manual parameters. These parameters are the basis of any gesture as they are used to convey meaning, similar to words in spoken languages. Facial expressions are used to enhance manual gestures by conveying the tone and mood of words, similar to the tone, pitch, emphasis, etc., of the voice in spoken languages.

Given a description of each of these parameters as they change in a sign language video sequence, it is possible to fully infer the sign language gestures performed in the video. This research focuses on accurately recognising and transcribing the sign language *manual* parameters in a sign language video. The recognition and transcription of facial expressions is left to future work.

A relatively large body of research exists on the recognition of sign language manual parameters from a monocular view, such as [10–19]. The ultimate goal is to simultaneously recognise and represent all four sign language manual parameters from a monocular view with a high accuracy and in real-time in a combined framework. This goal can be broken down into two main components: *hand retrieval* and *manual parameter representation and recognition*. The hand retrieval component aims to find and track the hands in a video sequence. Once located and tracked, the hands are then used by the manual parameter representation and recognition component to accurately isolate the hands, represent the hands in terms of relevant salient features, recognise the relevant manual parameters and represent them in a computer-readable sign language transcription notation.

Hand retrieval is comprised of the following sub-problems that need to be addressed: finding a means of accurately and interactively locating the hands;

devising an enhanced method of adaptive skin detection that more comprehensively and accurately highlights the hands regardless of shape and orientation; and finding a suitable means of accurately tracking both hands regardless of shape and orientation, which is robust to occlusions of the face and hands.

Manual parameter representation and recognition is comprised of the following sub-problems that need to be addressed: devising a suitable method of accurately recognising the orientation of the hands; devising a suitable method of accurately recognising the shape of the hands in a variety of orientations; and finding a non-limiting means of representing hand motions.

It should be noted that in all these cases, the strategies devised must operate in real-time; must be interactive and automatic, requiring minimal user input, if any; and must be robust to a range of skin tones, body types, gender, etc., to accommodate the wide ethnic diversity in South Africa.

In addressing the problem of sign language recognition the following assumptions are made:

- The camera is fixed and the signer is generally stationary. Also, the signer sits (or stands) at a relatively consistent distance from the camera.
- The entire upper body and head of the signer are visible in the video. Only the upper body is required to convey meaning in sign languages [20] and, as such, the input will mostly consist of the user's upper body.
- The signer faces the camera. Profile views or rotations of the body or head are not considered.
- When performing gestures, the signer does not move extra-ordinarily slow or fast.

The remainder of this paper provides a detailed discussion of related work in each of the sub-problems mentioned earlier in this paper, namely, skin detection (Sect. 2), hand tracking (Sect. 3), hand shape recognition (Sect. 4), and hand motion/gesture recognition (Sect. 5). The paper is summarised in Sect. 6. Altogether, this paper demonstrates the shortcomings in the literature, and highlights key techniques that can be used to improve future frameworks.

2 Skin Detection

Skin detection is a very well-established field of research in image processing. It has various applications in the field of human–computer interaction and has been applied extensively in sign language recognition research [21]. Applied to a monocular view, the aim of skin detection is to arrive at a binary adaptation of the input image in which pixels corresponding to skin-coloured objects are highlighted and pixels corresponding to any other objects are dimmed.

Despite the wide variations in skin colour across different ethnicities, the chromaticity of skin under fixed lighting conditions is limited to a relatively small region in the range of representative colours in any colour space [22, 23]. The implication of this is that there is a relatively small likelihood that non-skin objects in a given scene with fixed lighting will chromatically resemble skin. Therefore, many applications that attempt to automatically track humans or body parts use skin detection as a means of isolating objects of interest and of eliminating unwanted noise before applying tracking [24–29].

Several skin detection strategies have been proposed. Generally speaking, these strategies can be sub-divided into static, parametric, and non-parametric approaches [21]. The following subsections describe each of these approaches and provide prominent related work pertaining to each approach.

2.1 Static Skin Detection

The assumption made by static skin detection models is that the skin colour of any ethnicity can be constrained to a fixed region in a colour space. Various colour spaces have been used with static skin models including RGB, HSV, normalised RGB, and YCrCb. The most prominent static skin model in the RGB colour space is that of Kovac et al. [30] given by a binarisation function S that takes in the red, green, and blue components of a pixel denoted R, G, and B as input as follows:

$$S(R, G, B) = \begin{cases} \text{Skin} & \begin{array}{l} \text{if } R > 95, G > 40, B > 20, |R - G| > 15, \\ \max(R, G, B) - \min(R, G, B) > 15 \\ \text{and } R > B \text{ and } R > G \end{array} \\ \\ \text{Not Skin} & \text{Otherwise} \end{cases} \tag{1}$$

This function is specific to daylight conditions. Under different lighting conditions, different functions are used, since the skin region occupies different locations in any colour space under different lighting conditions.

One significant limitation of static skin detection is the lack of a method to infer a set of optimal skin detection rules given a training set. The means of inferring a static skin model is arbitrary. Training can therefore be very complex.

In order to address this limitation, Gomez and Morales [31] proposed a constructive induction algorithm to infer various possible skin detection rules in the normalised RGB colour space. Starting with the normalised RGB components $\frac{R}{R+B+G}$, $\frac{G}{R+B+G}$, and $\frac{B}{R+B+G}$ (where R, G, B are the RBG components of the pixel) and $\frac{1}{3}$ (corresponding to equal R, G, B components), the algorithm progressively constructs increasingly complex combinations of these expressions using the basic arithmetic operations $+$, $-$, $*$, $/$. Using this approach, they generated a large number of different possible static skin detection functions and selected the most effective. This approach partially addresses the lack of a definite method to infer static skin detection rules, but the method is computationally complex and there is no means to determine the optimality of the solution obtained.

All static skin detection models have one common feature: they are tailor-made for a specific set of images on which they are trained. Generally speaking, aside from the illumination of the scene, the nature and quality of skin in an image is dependent on a number of factors [32], including the characteristics of the camera with which the image is captured, ethnicity of the test subject(s), and test subject individuality such as gender, size, facial hair, etc. Assuming fixed lighting conditions, it is clear that a very large and comprehensive training set that is well-representative of these factors is required to be able to obtain a static skin model that generalises well.

It is challenging to obtain such a training set, and even when one is available, training can be time-consuming and complex. It is also very likely that the eventual model will fail under conditions that are unrepresented or under-represented in the training set. As such, static skin detection holds limited promise in a dynamic skin detection context.

The main advantage of static skin detection is its simplicity and fast processing speed in run-time [31, 32].

2.2 Parametric Skin Detection

Parametric skin detection is based on the assumption that the colour of skin pixels follows a fixed probability distribution. Most commonly, the underlying probability density is chosen as Gaussian:

$$p(\boldsymbol{c}|skin) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \cdot \exp\left(-\frac{1}{2}(\boldsymbol{c}-\boldsymbol{\mu})^{\mathbf{T}}\Sigma^{-1}(\boldsymbol{c}-\boldsymbol{\mu})\right), \tag{2}$$

where $\boldsymbol{c}$ is a colour vector $\boldsymbol{c} = \langle c_1, \dots, c_n \rangle$ from a set of n different channels of one or more colour spaces and p is parametrised by a mean vector $\boldsymbol{\mu}$ and covariance matrix Σ computed on a training set of skin pixels in the same channels and colour spaces as $\boldsymbol{c}$. A pixel is identified as skin if the probability density for the pixel's colour vector surpasses a given threshold. This identification rule corresponds to the thresholding function $\mathrm{S}(c)$ given below:

$$S(\boldsymbol{c}) = \begin{cases} \text{Skin} & \text{if } p(\boldsymbol{c}) \geq t_s \\ \text{Not Skin} & \text{if } p(\boldsymbol{c}) < t_s \end{cases} \tag{3}$$

The use of a colour vector $\boldsymbol{c}$ rather than a single colour channel provides greater flexibility in modelling the skin region and almost invariably results in better skin detection results [33–36]. Several studies have been carried out to compare the use of various colour spaces and channels for multivariate parametric skin detection [34, 35, 37]. While it is clear that the use of more than one component more often than not provides better results than the use of only a single component, the exact number of colour spaces and components appears to be arbitrary and specific to the data set used.

The use of a Gaussian distribution leads to a very compact model of the skin colour distribution in terms of only a small number of parameters. Very importantly, the method of training, though potentially computationally intensive, is explicit and defined and can be used to determine an optimal skin detection model. It can be used to accurately represent a complex skin region in multiple colour spaces in a single model. However, as in static skin detection, lighting variations remain a significant challenge. A single parametric model can only realistically incorporate very small variations in lighting before the skin region overlaps with non-skin regions in the colour space(s) used. Aside from lighting, a large training set that is well-representative of scene and subject variations is required to achieve a model that generalises well. As mentioned in the previous subsection, acquiring a training set that is representative of a comprehensive set of variations in images is challenging. It is likely that the skin model developed will fail under any varied conditions.

2.3 Non-parametric Skin Detection

Similar to parametric skin detection methods, non-parametric skin detection methods attempt to assign to a pixel a probability that specifies the likelihood of the pixel being skin. A mapping of the pixel onto the colours of skin and non-skin in a training set is used to determine the probability. Rather than modelling skin as a *region* in a colour space, non-parametric skin detection models skin as a set of colours. An unknown pixel is assigned a probability depending on how well its colour is represented in the training set.

This makes for a very flexible and dynamic representation of skin that does not necessarily have fixed boundaries, resulting in more accurate skin detection results than the previous two methods [38]. The probability can then be binarised as being skin or non-skin using a thresholding function S similar to that in Eq. (3).

Non-parametric skin detection is of two types. The first type makes use of a Bayes' classifier as a skin colour probability model. The model is trained on positive and negative samples of skin colour to obtain two probabilities $p(c|\text{skin})$ and $p(c|\neg\text{skin})$, the probabilities of observing a specific colour c, given that the colour is either skin or non-skin, respectively. These are then used in the following probability function p, which is an application of Bayes' rule, to determine the probability of observing skin given a specific colour c:

$$p(\text{skin}|c) = \frac{p(c|\text{skin})p(\text{skin})}{p(c|\text{skin})p(\text{skin}) + p(c|\neg\text{skin})p(\neg\text{skin})} \tag{4}$$

This approach has been used for skin detection in a variety of contexts [24, 39–41]. A range of colour spaces have been used, including the YCrCb colour space [39], a log-transformed RGB colour space [41], the YUV colour space [24], and others.

The second non-parametric skin detection approach makes use of a multidimensional histogram to represent the skin colour distribution. The colour space is quantised into bins, where each bin corresponds to a limited range of values in each colour component. Given a colour c that is located in a bin with histogram count $h(c)$, the conditional probability density $p(c|skin)$ is computed as:

$$p(c|skin) = \frac{h(c)}{h_{\text{total}}}, \tag{5}$$

where h_{total} is the sum total of all bin counts. This is a computationally efficient procedure that can yield accurate results given an appropriate training set. For this reason, this method has been used extensively for skin detection using a variety of colour spaces [29, 40, 42–46].

Generally speaking, non-parametric skin detection is also illumination-specific, like the previous two methods. It also, potentially, requires a large representative training set to generalise well. It is very important to limit the amount of noise in the training set used to compute the skin model, since even relatively small amounts of noise can significantly degrade the accuracy. Like parametric skin detection, training can be achieved to convergence, given a method of training is explicitly defined.

One adaptation that has been applied to histogram-based skin detection is to incorporate an online training procedure, made possible by the computational efficiency of the method. Specifically, many studies [29, 44, 47–49] incorporate a face detection component—commonly the Viola–Jones face detector [50]—and use a portion of the face region to compute a dynamic skin histogram of a specific user that adaptively represents the scene illumination, test subject individuality and ethnicity, and the characteristics of the camera used. If the region of the face used to compute the dynamic skin histogram is accurately segmented, any drawbacks relating to illumination sensitivity and noise sensitivity are circumvented and the skin model obtained is seamlessly adapted to any conditions observed. This method is henceforth referred to as adaptive histogram-based skin detection.

Using this specific approach, and assuming that the face of the intended user of the system is mostly visible, it is possible to achieve skin detection results similar or superior to those of other methods without the need for any prior training set at all, as in [29, 44, 47–49]. Figure 1 provides an example of skin detection using this approach.

This approach holds significant promise towards dynamic skin detection but currently has a number of limitations that need to be addressed. The skin colour of various parts of the hands of many users is chromatically different from that of their face. Also, even in cases when the user's face is chromatically close to the hands, the final skin image typically has a large number of holes. Finally, in some cases, skin and non-skin probabilities may be in close proximity, making it challenging to dynamically select a threshold that accurately and consistently separates the two.

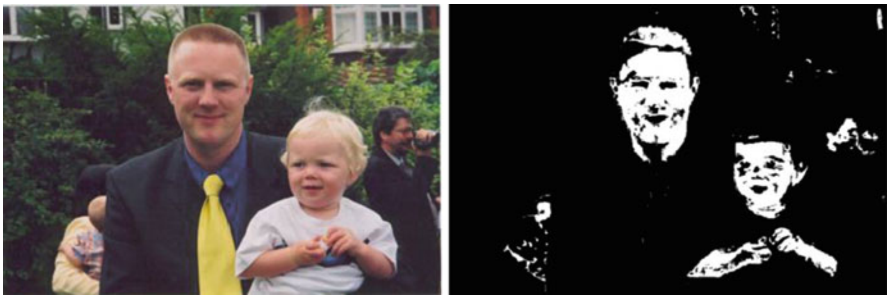


Fig. 1 Non-parametric skin detection using adaptive histogram-based detection: (left) Original image; (right) binary skin detected result [44]

3 Hand Tracking

Hand tracking has been investigated in some research studies and advances have been made in the field, but it remains a largely unsolved problem [6, 9, 51]. It has wide-ranging applications in human–computer interaction such as in virtual reality, robotics, games, and sign language recognition, among others [51].

General object tracking involves detecting/locating a specified object across an image sequence. The distinction between object detection and object tracking is that the former searches for the object in the entire frame in every frame in the sequence, whereas the latter makes use of contextual information from one or more previous frames in the sequence to limit the search space, resulting in greater computational efficiency [51].

Hand tracking is a specialised field of object tracking in which the object to be tracked is a hand: it is articulated and has many degrees of freedom and is therefore highly deformable. While rigid objects can be tracked by means of a continuous contextual search for their shape/contours, the hands cannot be tracked using this strategy if they are assumed to continuously move and change shape. Hand tracking is therefore notoriously challenging [6, 9, 51].

Hand tracking strategies can broadly be divided into two groups: those that track a single hand and those that track both hands simultaneously. Tracking both hands is a significantly more challenging tracking problem when the hands overlap with each other or the face, causing occlusions of the hand(s) or face [8]. In such cases, a strategy is required to resolve occlusions and distinguish the hands and face from each other. For this reason, studies that attempt to track both hands are very limited.

The following subsections describe studies within each of these hand tracking groups.

3.1 Approaches to Tracking a Single Hand

Rautaray and Agrawal [26] tracked the hands by means of skin detection and a statistical means of locating the hand in each frame. At the core of the method is the assumption that the hand is the largest object in the input. In each frame, a static skin detection model in the L*a*b* colour space is first used to detect objects of skin colour. Thereafter, connected components labelling [52] is used to find clusters of skin. Finally, the median of the skin clusters is used to infer the position of the hand.

Figure 2 demonstrates the tracking system in action. This approach can track the hand regardless of shape and orientation, but it is highly sensitive to skin-coloured noise regions in the frame and limits the freedom of the user by making stringent assumptions on the nature of the input.

A number of other similar studies [27, 53, 54] have made use of a similarly simple tracking strategy with the same limitations and comparable effectiveness.

Kölsch and Turk [25] introduced a new method of pruning the LK optical flow tracking approach [55] for hand tracking that they called the “flocks-of-features” method. The pruning method is derived from the flocking organisation of birds in flight. While each bird moves individually, it maintains proximity with other birds in the flock and the median of the flock. In terms of hand tracking, the hand is assumed to be a flock of skin-coloured KLT features [56]—also known as “good features to track”—which move individually while maintaining proximity with other features in the flock and the median of the hand as it moves.

A large number of features—100 or more—are computed on skin-coloured regions of the hand in the first frame. As the hand moves and changes shape in every subsequent frame, the positions of the feature points are updated using LK optical flow tracking, subject to the proximity constraints. Tracking is achieved by computing the centroid of the hand which is assumed to be the median of the feature points in each frame. Figure 3 demonstrates the system in action.

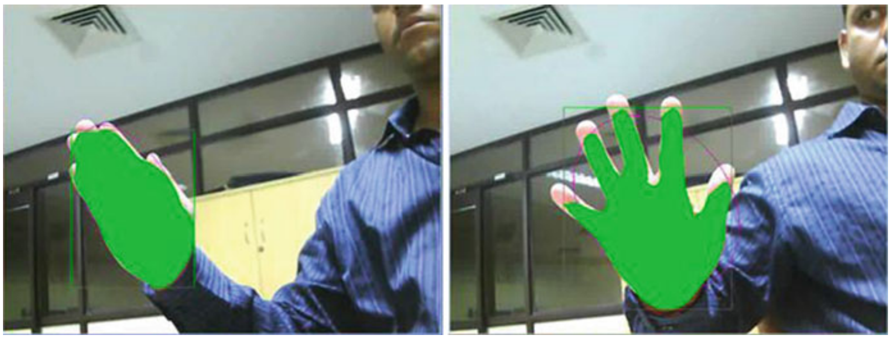


Fig. 2 The hand tracking system of Rautaray and Agrawal [26]

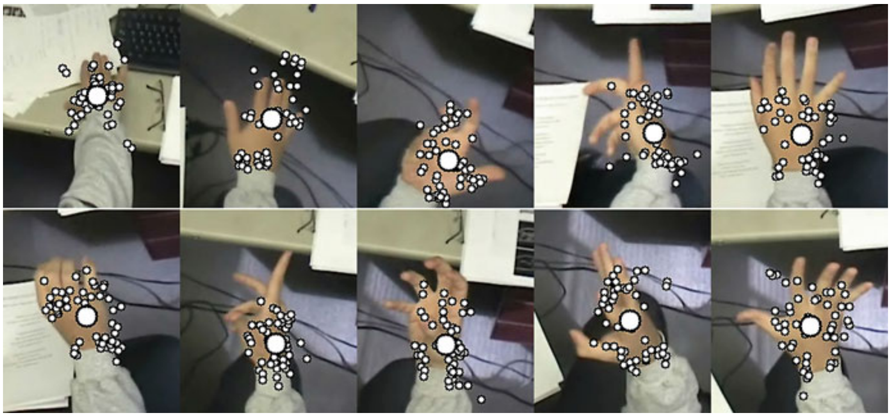


Fig. 3 The hand tracking system of Kölsch and Turk [25]

At the core of this approach is the assumption that the only skin-coloured object in the input is a single hand. It is very sensitive to any competing skin-coloured noise and strong edges in the background, the presence of which can cause incorrect relocations of feature points in their proximity and, ultimately, tracking failure.

A modification to this approach by Fogelton [28] proposed the application of this approach to a skin probability image rather than to an original grayscale equivalent of the input. This eliminates edges in the background that could otherwise interfere with tracking. The skin probability image is obtained by means of histogram back-projection of a sample of the hand in the HSV colour space onto an input image in the image sequence. A set of morphological operations is used to reduce noise regions in the probability image.

Visual results indicate that this modification significantly improves tracking performance. Figure 4 illustrates the modified algorithm in action. In the figure, the hand is passed over the face without causing a loss of tracking. This is mainly achieved by ensuring that the hand occupies the majority and centre of the input in the period over which the occlusion takes place, without which the face is likely to "steal" the tracking features.

The hand tracking strategies discussed up to this point have used a single cue, i.e., skin colour, to track the hand, making these strategies highly susceptible to skin-coloured noise in the background. A number of studies have investigated the use of additional cues to eliminate skin regions that are not part of the hand(s).

Li et al. [14] proposed a hand tracking system which provides significantly more freedom to the user than previous systems. The system combines a skin cue with a motion cue to limit skin-coloured background noise that could interfere with tracking. Assuming a stationary camera and a generally stationary user, a combination of the skin and motion cues effectively eliminates many stationary skin regions including the face and neck and any background skin-coloured noise.

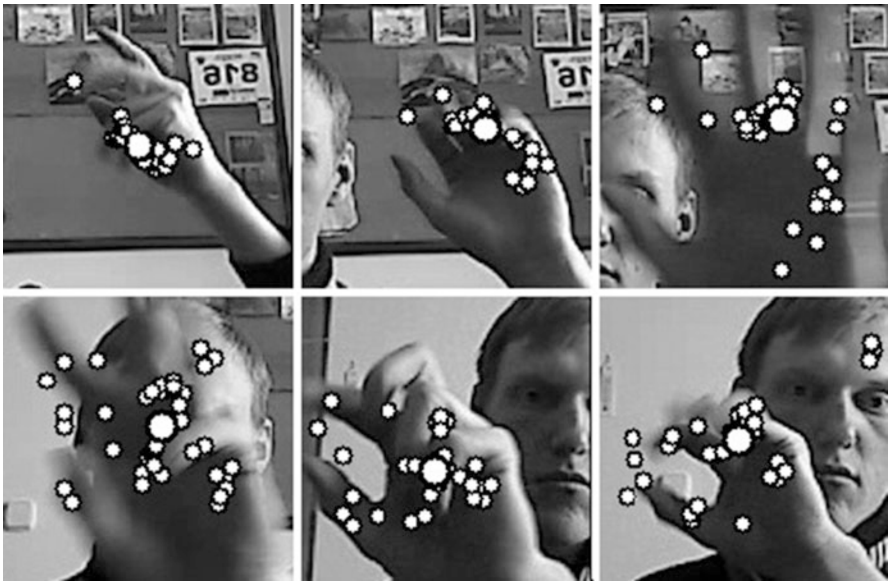


Fig. 4 The hand tracking system of Fogelton [28]

Depending on the motion highlighting method used, this can be an efficient and useful combination for tracking.

Skin detection is carried out by means of adaptive histogram-based skin detection in the HSV colour space. Motion detection is achieved with background subtraction using Gaussian mixture models (GMMs) to model the motion of each pixel in time. The CAMShift tracking algorithm is then used to track the hand which is assumed to be the most prominent moving skin region in the frame. Making use of a multi-cue tracking approach provides significantly more freedom to the user than single-cue approaches.

Asaari and Suandi [57] also combined motion features with colour features to more robustly highlight skin objects to track. A static skin detection model in the YCrCb colour space was used to highlight skin. As with others, the luminance channel was omitted to obtain greater robustness to illumination changes. Frame differencing, which is a simpler motion highlighting method, was used to highlight motion between subsequent frames. Finally, a Kalman filter was used to predict the location of the hand in every frame based on the skin and motion observations. The results indicate that the multi-cue system is capable of tracking a single hand in a relatively less constrained environment than systems that use only a single (skin) cue.

Liu and Zhang [58] combined texture features with colour features and used a particle filter to track a single hand. The local binary patterns (LBPs) operator [59] was used to model texture features, while a histogram-based skin detection strategy was used to highlight skin in the HSV colour space. The visual results indicate that the strategy provides relatively greater robustness to skin-coloured background regions. In the figure, the hand is passed over a small region of the neck without a tracking loss.

Other studies similarly combine cues and use a tracking strategy such as MeanShift, CAMShift, Kalman filters, or particle filters [8, 10, 60, 61] with similar results.

A manual sign language parameter recognition necessarily requires input from both hands simultaneously. Tracking systems that track a single hand at a time therefore have limited promise in this context, but it is clear that a multi-cue strategy is a key to providing greater user freedom.

3.2 Approaches to Tracking Both Hands

A smaller number of studies have attempted to track both hands simultaneously. Prominent examples are described.

Spruyt et al. [9] proposed a system in which motion, texture, and colour features were combined, and a particle filter was used to track the two hands. Edge detection was used to highlight texture features, and the skin detection method used was a parametric model in both the RGB and HSV colour spaces. The motion cue used was the GMM implementation by Zivkovic [62]. Visual results demonstrate accurate tracking of both hands on a complex background. The strategy does not address, and is sensitive to, occlusions of the hands with each other and with the face. Furthermore, the system operates at a processing speed of 8 frames per second (fps) which is below real-time.

Coogan et al. [63] also proposed a multi-cue approach that uses motion, colour, and location features to limit sources of noise. A static skin detection method was used, although details of this are unclear. Frame differencing was used to obtain motion information. An additional cue was obtained by analysing the optical flow of gray-level pixels in subsequent images, making the assumption that pixels move by relatively small distances from frame to frame. A Kalman filter was used to convert the motion, colour, and location observations into predictions of the hand locations in every frame.

The management of self-occlusions was limited to detecting them and keeping track of which skin blobs, i.e., hands and/or face, have occluded in any given frame. No strategy was proposed to resolve such occlusions when the respective blobs separate and emerge. Visual results of the system are provided in Fig. 5 and demonstrate effective tracking within the limits of the system. The system operates at a processing speed of 10 fps which is significantly less than real-time.

Wen and Zhan [46] proposed a system that uses a colour cue combined with a position forecasting model and the CAMShift tracking algorithm to track both hands. Skin detection was achieved by histogram back-projection using the Cr channel of the YCrCb colour space. A position forecasting model based on a recent location history of each hand was used to improve tracking and keep track of

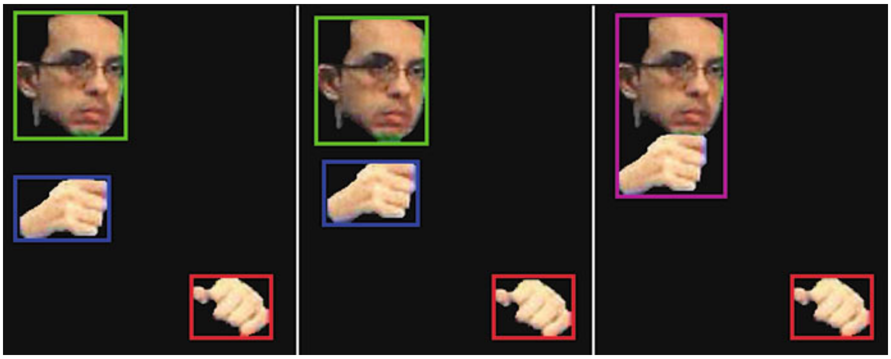


Fig. 5 The hand tracking system of Coogan et al. [63]

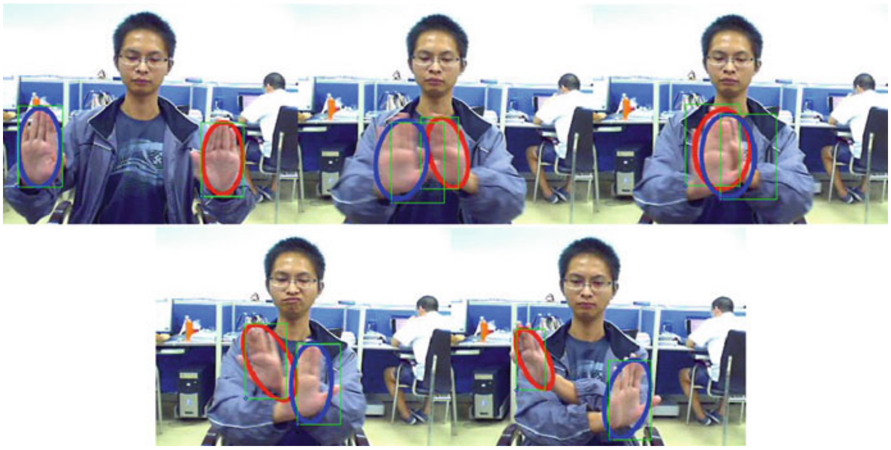


Fig. 6 The hand tracking system of Wen and Zhan [46]

occlusions as they took place. With this approach, occlusions that take place with one or both hands in continuous motion can be resolved. Figure 6 demonstrates the system in action. If the hands stop at the moment of occlusion, they can no longer be resolved. The system operates at a real-time speed of 25 fps on frames of size 320 × 240 pixels.

Argyros and Lourakis [24] proposed a system that uses data association to track objects. A non-parametric skin detection procedure was used in the form of a Bayes' classifier trained on a large training set of images. The YUV colour space was used, with the luminance (Y) component discarded. Thereafter, data association was used to track skin-coloured objects.

With this method, it is assumed that ellipses can approximate the shape, size, and location of highly deformable objects to be tracked very well. Therefore, objects are modelled as ellipse hypotheses, the parameters of which are continuously updated as new observations are obtained in the image sequence. Assuming that relatively small changes in the location and shape of objects take place between any two consecutive frames, it is possible to associate observations with existing hypotheses very well. A simple motion forecasting method is also used to predict the location of each blob based on its two most recent locations to avoid losing track.

Given a processing frame rate that is close enough to real-time and assuming that a user moves his/her hands in a natural way, this method can work very well. Very importantly, contrary to other methods, the method can keep track of objects during occlusions, even if the objects stop during the occlusion, with a reasonable assumption that at least a small part of each occluding object will always be visible, i.e., if one hand moves in front of the other hand, at least a small part of the occluded hand will still be visible.

Holden et al. [64] proposed and used a very similar hand tracking approach.

The exact implementation of this strategy by Argyros and Lourakis is not described, but is visually demonstrated to be accurate and able to track objects in frames of size 320 $\times$ 240 pixels at a real-time frame rate of 28 fps. Figure 7 demonstrates the system in action. This is therefore a very promising approach for the proposed sign language manual parameter recognition system. However, with only a high-level description of the system provided and no algorithmic and implementation-level details presented, it may be possible to replicate the tracking accuracy of the system to some extent, but it will be very challenging—if at all possible—to replicate the high processing speed obtained.

It is well-known that any given system can be implemented in many ways, each with an inherent computational complexity and efficiency [65]. A problem as simple as sorting a list can be achieved in several ways, each with an inherent computational efficiency and cost, towards achieving the same end result. The problem is, therefore, to devise a custom implementation of this tracking strategy that is efficient and accurate.

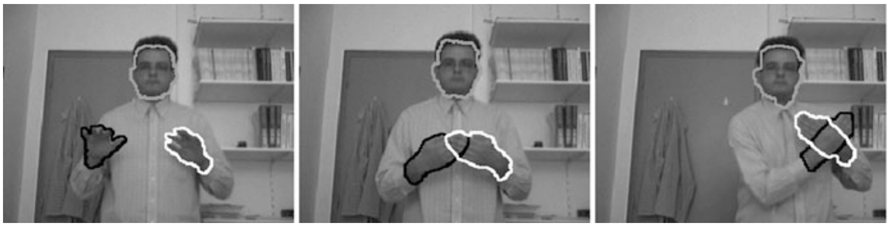


Fig. 7 The hand tracking system of Argyros and Lourakis [24]

4 Hand Shape and Finger-spelling Recognition

A large body of research exists on the recognition of hand shapes for a variety of sign languages from a monocular view. All of these studies assume that the hand is orientated frontal-upright in the plane of the camera view, i.e., a single orientation. Any rotations of the hand in any axis are treated as a misalignment of the hand. They are either excluded from consideration, or are corrected for by means of a normalisation procedure.

Also, the majority of these studies aim to recognise finger-spelling hand shapes specific to a sign language of choice. The majority of the studies focus on American Sign Language (ASL) finger-spelling recognition due to the relatively wider availability of ASL finger-spelling data sets. Finger-spelling gestures are generally not part of sign languages [66] and their use in these sign languages is very limited [67–69]. Regardless of this, it should be considered that the actual shapes being recognised in any specific study can be considered as arbitrary. To a great extent, if a given set of hand shapes can be accurately recognised, the recognition strategy can be extended to other sign language-specific shapes, although with a potentially different recognition success rate. For this reason, studies that perform hand shape recognition in a non-sign language context are not excluded from consideration and thus discussed in this section.

In each study, a fixed set of hand shapes are pre-selected and recognised. The pre-selection of a fixed number of shapes is well justified by the fact that each sign language has a fixed and finite number of distinct hand shapes. However finite, the potentially large number of possible shapes makes the collection of a completely comprehensive training data set very challenging. For this reason, every study limits the scope of recognition to a reasonable subset of hand shapes, with the promise of extension in the future.

Hand shape and finger-spelling recognition strategies have broadly been subdivided into two categories in the literature [70, 71] according to the approach used to classify specific hand features into hand shape classes. The first approach, called the "rule-based" approach (also sometimes referred to as the "template-matching" approach), makes use of a set of pre-defined rules to achieve classification. The second approach, called the "machine learning" approach, uses machine learning techniques to achieve classification.

The following subsections describe studies that use the rule-based and machine learning approaches. In each case, a selection of prominent studies is described along with their relevant pre-processing steps, to provide a basis of understanding, with further references provided to the interested reader.

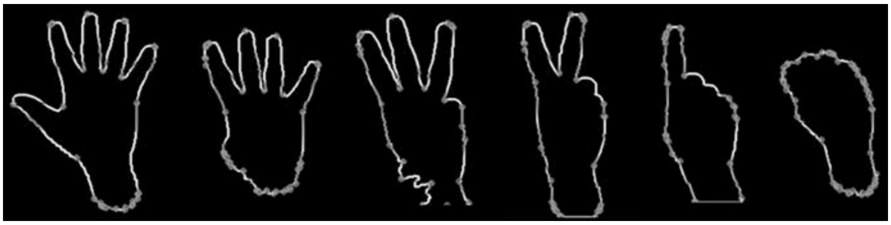


Fig. 8 Contours of hand shapes recognised by Liu and Kehtarnavaz's system [10]

4.1 Rule-Based Approaches

Liu and Kehtarnavaz [10] proposed a system for the recognition of six arbitrary finger-spelling hand shapes that were specifically categorised on the number of extended fingers presented. Although the main focus of the study was the use of stereo images for this task, they compared the use of a monocular view with a stereo view. This discussion focuses on the results obtained from a monocular view. The hand was tracked using the CAMShift tracking algorithm based on a skin detected image obtained by means of a multi-variate Gaussian distribution in the YCrCb colour space. A region growing approach [72] was applied to the centre of the CAMShift tracking window in order to segment the hand region more accurately. This was followed by the use of an unspecified set of morphological operations to highlight the outer-most contour of the hand. Figure 8 illustrates example contours of the six hand shapes considered. It is seen that the shapes are all in a frontal-upright orientation in the plane of the camera view.

To recognise the hand shapes, a convex hull was fitted onto the hand contour [73] and a set of pre-defined rules were applied to the size and defects of the convex hull to infer the hand shape. Fifty images per hand shape were used to test the system, although the number of, and diversity in, test subjects is not clear. Despite the simplicity of the setup, the accuracies obtained were relatively low, the lowest being 55% for hand shape "five," and the highest being 67% for hand shape "zero."

Kang et al. [11] proposed a system for the recognition of 12 arbitrary hand shapes very similar to those used by Liu and Kehtarnavaz. It was assumed that the input would consist of only an upright hand on a simple background. A static skin detection model in the chrominance components of the YUV colour space was applied to each hand image. Thereafter, connected components labelling was used to locate the largest contour in the skin image, which is the hand. The opening morphological operation was used to isolate the extended fingers in the resulting image. Finally, a second round of connected components labelling was used to identify and count the number of contours, i.e., fingers, that remained in the resulting image. Classification into one of the 12 hand shape classes was achieved by simply counting the number of fingers in the image. An unclear number of images from five subjects that are not described were used to test the system, but the system is claimed to correctly classify in "99% of the cases"[11].

Schreer and Ngongang [12] proposed a strategy to recognise 13 ASL hand shapes. The strategy assumes the use of binary images consisting of the hand to be recognised. Using the contour image, a template distance map is produced. This is done by computing the distance of the tangent at each point on the contour to the corresponding contour on the opposite side of the hand. Plotting the distances of all these points produces a template distance map that characterises each hand shape uniquely. Classification is achieved by computing the distances between peaks in the distance map and using a set of rules to assign the observed distances to one of the 13 ASL hand shapes. A "satisfactory" classification accuracy is claimed, and only a small sample of visual results are provided.

Shimada et al. [74] used a similar rule-based approach with a similar outcome.

In [75], the authors proposed a slightly different model for recognising American Sign Language (ASL) finger spelling in online videos. Sourcing data from this unconventional source (online videos) presented numerous challenges such as varied lighting conditions, background noise, camera angle, low pixel resolutions. The authors had to perform a number of pre-processing on the data. Test were conducted using local encode–decode and CTC and the results showed that accuracy increased with number of frames per second in the video, though the overall accuracy for both methods was less than 45%, with CTC scoring higher in most of the tests.

4.2 *Machine Learning Approaches*

Shi et al. [76] presented a benchmarking dataset for Arabic Sign Language (ArSL) alongside a sign language recognition algorithm which incorporates segmentation (using depth and position), hand shape sequencing based on histogram of oriented gradients (HOG) and principal components analysis, as well as body motion description and classification. Finally, the authors used both canonical correlation and random forests to test the approach and obtained a classification accuracy of 55%.

In a similar work, Alzohairi et al. [77] also worked on ArSL. Due to the similarity between signs in ArSL, most imaged-based recognition software have low accuracy. The authors compared a number of classifiers and reported that the HOG gave the highest accuracy for 27 of the 30 ArSL signs, while for the last 3 signs, the edge histogram descriptor (EHD) was better.

In the work of Kang et al. [78], a method for distinguishing between hands and objects using a two-stage random decision forest (RDF) was proposed. Compared to the traditional RDF and fully convolutional network (FCN), the proposed approach showed comparable results with shorter processing time.

Aryanie and Heryadi [13] proposed a strategy to recognise 10 ASL finger-spelling hand shapes. The strategy makes use of pre-segmented images of the hand and, as such, no skin detection or hand tracking strategy is required or used. The hand shapes are characterised by means of 16-bin histograms on each channel of the RGB colour space, resulting in a combined histogram of 48-bins which is taken to be a 48-dimensional feature vector.

A k-nearest neighbours (kNNs) classifier is used to classify an image into one of the 10 ASL hand shape classes. The classifier was trained on a data set consisting of between 517 and 555 samples per hand shape captured by 5 Caucasian signers. The best model was shown to achieve a near-perfect accuracy of 99.6% across all shapes and signers. The use of RGB histograms to characterise hand shapes implies very strong colour-dependency of the classifier. The same hand shapes performed by signers of a different skin colour would have very different feature vectors and would have to be modelled as separate hand shape classes altogether. The strategy is, therefore, very signer-dependent.

Saremi et al. [79] proposed a hybrid of multi-objective particle swamp optimisation (MOPSO) and evolutionary population dynamics (EPD), which was applied to determine the Pareto optimal front of hand postures. Tests were done using data from different datasets and the results showed that the hybrid model proposed was better than the classical MOPSO.

Cai et al. [80] proposed a weakly supervised model which relies on the depth information available in images taken using RGB-D cameras. This helps to overcome the challenges of scarce annotated 3D hand-pose datasets and long training times. Using the depth information, the authors created a synthesised 3D dataset and used a depth regulariser to compensate for the absence of a true ground-truth. Results of experiments performed on both synthesised and real hand-image datasets show that the proposed approach was at par in most tests and even out-performed the compared state-of-the-art methods in some instances.

Li et al. [14] proposed a hand shape recognition system to recognise 10 SASL hand shapes. As with other studies, these hand shapes are orientated in a frontal-upright position in the plane of the camera view. Li et al.'s skin detection and hand tracking strategy was described in Sect. 3.1. Once the hand is tracked and isolated, connected components labelling (CCL) [52] is used to locate the largest contour in the CAMShift tracking window, which represents the hand. The contour image is resized to a constant size of 20×30 pixels. The resulting image is flattened to obtain a 600-dimensional feature vector. Classification is achieved by means of support vector machines (SVMs).

Training was carried out using 40 examples captured from each of two test subjects, and testing, using 15 examples captured from five different test subjects. The test subjects were of different gender and skin tone. The system achieved a high overall recognition rate of 81%.

de Paula Neto et al. [15] proposed a recognition strategy for 18 Brazilian Sign Language finger-spelling hand shapes. The strategy operates on binary images of the hand shapes orientated in a frontal-upright position facing the camera. Each of the images is of a fixed size of 150×100 pixels. Features are extracted from each binary hand image by applying a custom-defined zoning statistical operation to the image. This involves dividing the image into a pre-defined set of blocks and taking the ratio of the white and black pixels in each block as the feature value for that block. In effect, these features mainly represent the outer contours of each hand,

and the inner texture of the hand to a smaller extent whenever such contours are available.

The features are used as input to a multi-layer perceptron (MLP) that classifies the features into a hand shape class. Having been trained on 15 images of each hand shape, and tested on 40 images of each hand shape, the system achieved a high average accuracy of 95%.

Hu [81] proposed a similar system for ASL 24 finger-spelling letters. Several feature descriptors were compared with SVMs used for classification. High accuracies of 85% and above were achieved for every letter.

While the feature description method used differs slightly from study to study, the majority of machine learning-based hand shape recognition studies [16–19] make use of features pertaining to the outer contours of the hand to characterise hand shapes, similar to Li et al. and de Paula Neto et al. In all of these cases, a relatively larger vocabulary can be recognised at a higher accuracy compared to rule-based approaches; this is due to the use of machine learning techniques to automatically infer a model. However, the use of only the outer contour to characterise the hand shapes limits the possible visual variations that can be recognised.

In contrast, Maqueda et al. [71] and Bastos et al. [82] recently used texture-based feature descriptors to recognise much larger vocabulary sizes of 24 and 40 hand shapes, respectively, at very high accuracies.

Maqueda et al. proposed a new LBP feature descriptor that they refer to as the "local binary sub-patterns" (LBsP) feature descriptor. Like the original LBP, the LBsP is a representation of the local and global texture of objects in an image. Using the LBPs and images of segmented hands, they were able to achieve very high accuracy classification—97.9%—of 24 hand shapes using the SVM classification technique. Bastos et al. used two different texture descriptors—the HOG and the Zernike Invariant Moments (ZIM). Using MLPs, they achieved 96.77% accuracy with the HOG and 86.62% with the ZIM on a significantly larger set of 40 hand shapes—almost twice as many as Maqueda et al.

A static hand gesture recognition for low-powered mobile devices, which combines HOG with LBPs was developed in [83]. The model is vision-based and uses the camera of the mobile device to capture input. Texture extraction was done using LBPs, contour information using HOG, and AdaBoost was used for the training process. AdaBoost is particularly suitable for a low-powered mobile device context since it has a low energy footprint. The model resulted in a recognition accuracy of 92% at a distance of 0.75 m or less from the camera.

5 Hand Motion/Gesture Recognition

Hand motion recognition has generally been referred to as gesture recognition in the literature [51]. All of the studies that attempt hand motion recognition do so for a fixed set of pre-defined hand motions, with almost all studies disregarding the hand shape. A specific hand motion is then taken as representing a specific word or phrase in a specific sign language. In all of these cases, words or phrases that have distinct hand motions are selected and recognised. Many of these studies obviously fail to discriminate between two or more words or phrases that have the same hand motion, but different hand shape and/or orientation.

Selected studies are described below, with additional references provided for the interested reader.

Yang et al. [84] proposed a system to recognise 18 hand motions including finger-drawings of shapes, namely, circle, triangle, and square, the letters “a”–“e” and numbers 1–5, as well as the motions of grasping, throwing, and waving. Static skin detection on the hue component of the HSV colour space was used to highlight skin, and tracking was achieved by simply assuming that the hand is the only skin-coloured object in the frame at any time. Several features of the hand, including its position relative to the centre of mass of the gesture, its velocity, and its size are collated across the gesture sequence into a time-dependent feature vector. Hidden Markov models (HMMs) are trained and used to recognise 18 unique hand motions with a very high average accuracy of 96%.

Matsuo et al. [85] proposed a similar but improved system. The improvement that they proposed pertains to the method of generating HMM models. They proposed a method of generating multiple HMM models for each recognised hand motion and selecting the simplest and most accurate model in each case. Improved accuracies are claimed on a set of 20 unique hand motions.

Holden et al. [64] proposed a similar system to recognise a set of 21 Australian Sign Language words. Parametric skin detection in the RGB colour space was used to highlight skin in the image. A tracking approach based on data association was used to locate and track the hands and face across the image sequence. In each frame, a set of features that were thought to characterise the words to be recognised were computed and accumulated. These included: the position of the hands and face relative to each other, the size of each of the hands and face in the frame, and the angle of the hands and face relative to each other. HMMs were trained to recognise the 21 words based on these features. A very high accuracy of 99% was claimed, although the exact testing procedure is not clear.

Many other similar studies exist [61, 86–88], with the majority of these studies making use of a custom set of features as input to HMMs or other classifiers to recognise a fixed set of hand motions.

The distinct hand motions that can be performed in sign languages are numerous. Words/phrases are performed in specific locations around the signer, and the hand motion trajectories taken towards those locations are also varied. The unique hand motion possibilities are therefore very numerous. As stated by Vogler and Metaxas [89], there is a “combinatorial explosion” when representing and recognising gestures holistically.

Therefore, whereas the use of a fixed classifier to recognise a finite set of classes is justified in the case of hand shapes and hand orientations, its use leaves a lot to be desired when applied to hand motions. It is highly unlikely that every, or even many, possible unique hand motions can be effectively represented, modelled, and recognised in any single classifier or system. This severely limits the vocabulary

Table 1 Taxonomy of related literature

Category	Reference	Strengths	Weaknesses
Static Skin detection	[21–23, 30–32]	Simple and fast processing speed (run-time)	1. Affected by lighting conditions
			2. Authors used arbitrary skin detection rules
			3. Optimal solution difficult to find
Parametric skin detection	[33–37]	The use of multiple components produced better results than when single component(s) were used	1. Variations in lighting conditions remain a significant challenge
			2. Requires large data set in order to generalise well
Non-parametric skin detection	[24, 29, 38–50]	1. Computationally efficient and yields accurate results if given appropriate training set(s)	Similar to those of the Parametric approach
		2. Dynamic skin histogram provides immunity to the effects of illumination changes	
		3. Skin detection can be done without the need for prior training	
Hand tracking			
Single hand, single hand tracking with multi-cue	[8, 10, 14, 25–28, 51, 53, 54, 57, 58, 60, 61]	1. The use of motion cues provides significantly more freedom to the users when compared with other systems	1. Very sensitive to any competing noise from skin-colouration or background
		2. A motion cue also minimises interference from stationary skin regions such as face and neck	2. The hand must be centred as well as occupy a large portion of the image frame
		3. Approaches that use multi-cue are more robust than those with single cue	3. Systems that utilise manual sign language parameters for recognition usually require input from both hands simultaneously; thus tracking only a single hand at a time is not ideal in such contexts

(continued)

Table 1 (continued)

Category	Reference	Strengths	Weaknesses
Track both hands simultaneously	[8, 9, 24, 46, 62, 63]	1. Can achieve high accuracy if there is no occlusion of hands	1. Most do not cater for hand occlusions, with the exception of [24]
		2. Tracking both hands has more realistic applications as most sign languages require both hands for communication	2. Most of these techniques require the hands to be in an upright position and on a simple background
			3. Most of these approaches are slow (running at 8–10 fps), hence not suitable for use in real-time applications
Hand shape recognition			
Finger-spelling recognition	[10–12, 70, 74, 81]	Due to the relative simplicity of proposed solutions, the computational resources required for implementation are usually low	1. The proposed approaches require the hand to be perfectly aligned and in a frontal-upright orientation
			2. In many environs, finger-spelling gestures are generally not considered a part of sign languages
			3. Most of these approaches focused on recognising specific finger-spelling shapes for specific sign languages, the majority being ASL
Rule-based or template-matching	[10, 13, 67–69, 72, 73, 75]	1. Most applied a relatively simple setup	1. Highly dependent on skin colour
		2. High prediction accuracy was reported for certain hand shapes	2. They are skin-specific in that they require different models to be built when trying to recognise the same signs done by signers with varied skin types or colour

(continued)

Table 1 (continued)

Category	Reference	Strengths	Weaknesses
Machine learning	[14, 15, 71, 76–80, 82–85]	Relatively high recognition accuracy (average 92%) in most cases	1. A few of the methods require frontal orientation of the hands
			2. They are mostly sign language specific, i.e., they can only recognise the language used in the training sets, e.g., Brazilian sign language or SASL or Australian sign language only, and not all or a mix of languages
Hand motion or gesture recognition	[61, 64, 86–88, 90–92]	1. High accuracy (99%) was reported	1. Only small (niche) data sets were considered
		2. Most employed a compact model to represent hand motions	2. A few had a cumbersome pre-processing step, requiring a re-representation/mapping of gestures as blocks of frames prior to usage

size, and hence the effectiveness and usability, of such strategies. A method of representing hand motions in terms of more fundamental features of these motions is required.

To this end, rather than viewing a hand motion as a single unit, Nel [90] proposed that hand motion be decomposed into, and represented as, a sequence of hand locations in time. This representation may be considered an indirect representation of any given motion, but it is completely flexible and can be used to represent virtually any hand motion, provided the hands can be accurately tracked. It may also be considered as a very cumbersome and verbose representation of any given motion, but this can be reduced to a very compact representation using a key-framing approach similar to that used in [91, 92] for human movement representation.

A summary of all of the surveyed literature is provided in Table 1.

6 Summary and Conclusions

This paper provided a literature review in the fields of skin detection, hand tracking, hand shape recognition, and hand motion recognition.

Skin detection was observed to be sub-divided into three main categories, namely, static, parametric, and non-parametric methods. The strengths and weaknesses of each method were detailed. The discussion showed that non-parametric methods show the most promise in terms of skin detection accuracy, providing both model flexibility like static methods and training simplicity like parametric methods. They also do not appear to be sensitive to the specific colour space used, although many studies indicate a beneficial effect from the use of chrominance information without luminance information.

Of the non-parametric methods, the use of histogram back-projection of a sample of the user's face onto the image was shown to provide the ability to dynamically adapt to environmental conditions such as user skin tone, lighting, etc. However, this method, in its current form, has shortcomings which limit its ability to highlight the hand(s) in a variety of orientations, as is required in this research. Therefore, this method will be adopted and modified in this regard in this research.

It was also shown that skin detection is almost always used as an initial object highlighting step before hand tracking is carried out. Once skin detection is applied, tracking is carried out on skin-coloured objects. This justifies the use of skin detection as the initial step in this research.

Hand tracking studies were divided into those that track only a single hand—the majority—and those that track both hands simultaneously—a smaller number of studies. Approaches to tracking only a single hand do not apply to this research. Of the approaches to tracking both hands, the data association approach is the only one that addresses the resolution of occlusions between the hands and face. Therefore, this approach is selected as the tracking approach in this research. A custom implementation of this approach will be devised and evaluated.

It was also seen that the use of only a skin cue in tracking causes the hand tracker to be susceptible to sources of skin-coloured noise in the background. It is crucial to include one or more additional cues to isolate objects to be tracked from noise. In this regard, the use of a motion cue was a popular choice due to its accuracy and computational efficiency. The motion detection method proposed by Zivkovic [62] was used by Spruyt et al. [9] with success. Based on this, this research makes use of a combination of skin and motion cues, with the motion detection method used being that of Zivkovic.

In terms of hand shape recognition, it was shown that all existing studies only consider hand shapes in a single orientation of the hand—frontal-upright and in the plane of the camera view. This strongly justifies the objective of this research to recognise hand shapes in a variety of orientations.

Hand shape recognition approaches were divided into rule-based and machine learning approaches, with machine learning approaches being more advantageous than rule-based approaches. The majority of studies that use machine learning approaches use features pertaining to the outer contour of the hand for classification. A smaller number of studies use texture features of the hand with significantly more success. The HOG feature descriptor showed special promise in this regard. Given the challenging nature of the two classification tasks undertaken in this research, i.e., recognising hand shapes in a variety of orientations and recognising the orientation of the hands, it was resolved to make use of texture features, specifically the HOG feature descriptor, in this research.

In terms of hand motion recognition, it was shown that the majority of studies devise and use a custom set of features pertaining to the location and motion of the hands in a video sequence, and mostly use HMMs to recognise a relatively small set of pre-defined hand motions. A discussion came to the conclusion that this approach is very limited, given the large range of possibilities for hand motions and the limited size and accuracy of any HMM classifier that can possibly be created. A more feasible and flexible method is to represent hand motions as a series of hand locations in time. This approach could be considered for future research work. It was also stated that hand orientation recognition has never been attempted in the literature. This can also form the basis for future work with particular focus on building a framework for manual parameter recognition and transcription of sign language.

Acknowledgment This research is supported in part by the National Research Foundation of South Africa (UNIQUE GRANT NO: 105670).

References

1. Deaf Federation of South Africa (2016). Available at http://www.deafsa.co.za/index-1.html
2. Statistics South Africa, Profile of persons with disabilities in South Africa, in *Census 2011* (2011)
3. W. Stokoe, Sign language structure, an outline of the visual communication systems of the American Deaf. J. Deaf. Stud. Deaf. Educ. **10**(1), 3–37 (2005)
4. S. Prillwitz, Hamburg Zentrum für Deutsche Gebärdensprache und Kommunikation Gehörloser, *HamNoSys: Version 2.0; Hamburg Notation System for Sign Languages; An Introductory Guide* (Signum-Verlag, Hamburg, 1989)
5. V. Sutton, *Lessons in Sign Writing* (SignWriting, La Jolla, 1995)
6. R. Wang, S. Paris, J. Popović, 6D hands: markerless hand-tracking for computer aided design, in *Proceedings 24th Annual ACM Symposium on User Interface Software and Technology* (ACM, New York, 2011), pp. 549–558
7. F. Jiang, J. Ren, C. Lee, W. Shi, S. Liu, D. Zhao, Spatial and temporal pyramid-based real-time gesture recognition. J. Real-Time Image Process., **13**, 1–13 (2016)
8. H.-S. Yeo, B.-G. Lee, H. Lim, Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware. Multimed. Tools Appl. **74**(8), 2687–2715 (2015)
9. V. Spruyt, A. Ledda, S. Geerts, Real-time multi-colourspace hand segmentation, in *Proceedings 17th IEEE International Conference on Image processing* (IEEE, Piscataway, 2010), pp. 3117–3120
10. K. Liu, N. Kehtarnavaz, Real-time robust vision-based hand gesture recognition using stereo images. J. Real-Time Image Process. **11**(1), 201–209 (2016)

11. S.K. Kang, M.Y. Nam, P.K. Rhee, Color based hand and finger detection technology for user interaction, in *Proceedings 2008 International Conference on Convergence and Hybrid Information Technology* (IEEE, Piscataway, 2008), pp. 229–236
12. O. Schreer, S. Ngongang, Real-time gesture recognition in advanced video communication services, in *Proceedings 14th International Conference on Image Analysis and Processing* (IEEE, Piscataway, 2007), pp. 253–258
13. D. Aryanie, Y. Heryadi, American sign language-based finger-spelling recognition using *k*-nearest neighbors classifier, in *3rd International Conference on Information and Communication Technology* (IEEE, Piscataway, 2015), pp. 533–536
14. P. Li, M. Ghaziasgar, J. Connan, Hand shape recognition and estimation for South African sign language, in *Proceedings South African Telecommunication Networks and Applications Conference* (2011), pp. 344–349
15. F.M. de Paula Neto, L.F. Cambuim, R.M. Macieira, T.B. Ludermir, C. Zanchettin, E.N. Barros, Extreme learning machine for real time recognition of Brazilian sign language, in *Proceedings 2015 IEEE International Conference on Systems, Man, and Cybernetics* (IEEE, Piscataway, 2015), pp. 1464–1469
16. Y. Sato, M. Saito, H. Koike, Real-time input of 3D pose and gestures of a user's hand and its applications for HCI, in *Proceedings Virtual Reality* (IEEE, Piscataway, 2001), pp. 79–86
17. S. Singh, P. Bharti, D. Kumar, Sign language to number by neural network. Int. J. Comput. Appl. **40**(10), 38–45 (2012)
18. V.S. Kulkarni, S. Lokhande, Appearance based recognition of American sign language using gesture segmentation. Int. J. Comput. Sci. Eng. **2**(03), 560–565 (2010)
19. J.J. Phu, Y.H. Tay, Computer vision based hand gesture recognition using artificial neural network. Technical report, Faculty of Information and Communication Technology, University Tunku Abdul Rahman, Malaysia (2006)
20. M. Filhol, Zebedee: a lexical description model for sign language synthesis. Technical report, LIMSI, National Centre for Scientific Research, Orsay, Paris (2009)
21. P. Kakumanu, S. Makrogiannis, N. Bourbakis, A survey of skin-color modeling and detection methods. Pattern Recogn. **40**(3), 1106–1122 (2007)
22. J.M. Chaves-González, M.A. Vega-Rodríguez, J.A. Gómez-Pulido, J.M. Sánchez-Pérez, Detecting skin in face recognition systems: a colour spaces study. Digital Signal Process. **20**(3), 806–823 (2010)
23. M.-H. Yang, D.J. Kriegman, N.Ahuja, Detecting faces in images: a survey. IEEE Trans. Pattern Anal. Mach. Intell. **24**(1), 34–58 (2002)
24. A.A. Argyros, M.I. Lourakis, Real-time tracking of multiple skin-colored objects with a possibly moving camera, in *Computer Vision-ECCV 2004* (Springer, Berlin, 2004), pp. 368–379
25. M. Kölsch, M. Turk, Flocks of features for tracking articulated objects, in *Real-Time Vision for Human-Computer Interaction* (Springer, Boston, 2005), pp. 67–83
26. S.S. Rautaray, A. Agrawal, A real time hand tracking system for interactive applications. Int. J. Comput. Appl. **18**(6), 28–33 (2011)
27. M. Elmezain, A. Al-Hamadi, J. Appenrodt, B. Michaelis, A hidden Markov model-based continuous gesture recognition system for hand motion trajectory, in *Proceedings 19th International Conference on Pattern Recognition* (IEEE, Piscataway, 2008), pp. 1–4
28. A. Fogelton, Real-time hand tracking using modificated flocks of features algorithm. Inf. Technol. Bull. ACM Slov. **3**(2), 37–41 (2011). Special Section on Student Research in Informatics and Information Technologies
29. I. Achmed, I.M. Venter, P. Eisert, A framework for independent hand tracking in unconstrained environments, in *Proceedings Southern African Telecommunication Networks and Applications Conference, George* (2012), pp. 159–164
30. J. Kovac, P. Peer, F. Solina, Human skin color clustering for face detection, in *Proceedings 2003 EUROCON, Ljubljana*, vol. 2 (IEEE, Piscataway, 2003)
31. G. Gomez, E. Morales, Automatic feature construction and a simple rule induction algorithm for skin detection, in *Proceedings of the ICML Workshop on Machine Learning in Computer Vision* (2002), pp. 31–38

32. M.R. Mahmoodi, S.M. Sayedi, A comprehensive survey on human skin detection. Int. J. Image Graph. Signal Process. **5**, 1–35 (2016)
33. D. Chen, G. Li, Y. Sun, J. Kong, G. Jiang, H. Tang, Z. Ju, H. Yu, H. Liu, An interactive image segmentation method in hand gesture recognition. Sensors **17**(2), 253 (2017)
34. T.S. Caetano, S.D. Olabarriaga, D.A.C. Barone, Performance evaluation of single and multiple-Gaussian models for skin color modeling, in *Proceedings XV Brazilian Symposium on Computer Graphics and Image Processing* (IEEE, Piscataway, 2002), pp. 275–282
35. J.-C. Terrillon, M.N. Shirazi, H. Fukamachi, S. Akamatsu, Comparative performance of different skin chrominance models and chrominance spaces for the automatic detection of human faces in color images, in *Fourth IEEE International Conference on Automatic Face and Gesture Recognition, 2000. Proceedings* (IEEE, Piscataway, 2000), pp. 54–61
36. J.Y. Lee, S.I. Yoo, An elliptical boundary model for skin color detection, in *Proceedings 2002 International Conference on Imaging Science, Systems, and Technology, Las Vegas, Nevada* (CSREA Press, Athens, 2002)
37. S. Ketenci, B. Gencturk, Performance analysis in common color spaces of 2D Gaussian color model for skin segmentation, in *Proceedings 2013 EUROCON* (IEEE, Piscataway, 2013), pp. 1653–1657
38. V. Vezhnevets, V. Sazonov, A. Andreeva, A survey on pixel-based skin color detection techniques, in *Proceedings of the GraphiCon, Moscow*, vol. 3 (2003), pp. 85–92
39. D. Chai, S.L. Phung, A. Bouzerdoum, Skin color detection for face localization in human-machine communications, in *Sixth International Symposium on Signal Processing and its Applications*, vol. 1 (IEEE, Piscataway, 2001), pp. 343–346
40. J. Brand, J.S. Mason, A comparative assessment of three approaches to pixel-level human skin-detection, in *Proceedings of 15th International Conference on Pattern Recognition*, vol. 1 (IEEE, Piscataway, 2000), pp. 1056–1059
41. D.A. Forsyth, M. Fleck, C. Bregler, Finding naked people, in *Proceedings European Conference on Computer Vision* (1996), pp. 593–602
42. M.J. Jones, J.M. Rehg, Statistical color models with application to skin detection. Int. J. Comput. Vis. **46**(1), 81–96 (2002)
43. B.D. Zarit, B.J. Super, F.K. Quek, Comparison of five color models in skin pixel classification, in *Proceedings International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems* (IEEE, Piscataway, 1999), pp. 58–63
44. M.J. Taylor, T. Morris, Adaptive skin segmentation via feature-based face detection, in *Proceedings SPIE Photonics Europe* (International Society for Optics and Photonics, Bellingham, 2014), pp. 91390P–1–12
45. A. Albiol, L. Torres, E. J. Delp, Optimum color spaces for skin detection, in *ICIP*, vol. 1 (2001), pp. 122–124
46. J. Wen, Y. Zhan, Vision-based two hand detection and tracking, in *Proceedings 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human* (ACM, New York, 2009), pp. 1253–1258
47. Q. Liu, G.-Z. Peng, A robust skin color based face detection algorithm, in *Proceedings 2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics*, vol. 2 (IEEE, Piscataway, 2010), pp. 525–528
48. L. Nanni, A. Lumini, F. Dominio, P. Zanuttigh, Effective and precise face detection based on color and depth data. Appl. Comput. Inform. **10**(1), 1–13 (2014)
49. T.-W. Yoo, I.-S. Oh, A fast algorithm for tracking human faces based on chromatic histograms. Pattern Recogn. Lett. **20**(10), 967–978 (1999)
50. P. Viola, M.J. Jones, Robust real-time face detection. Int. J. Comput. Vis. **57**(2), 137–154 (2004)
51. S.S. Rautaray, A. Agrawal, Vision based hand gesture recognition for human computer interaction: a survey. Artif. Intell. Rev. **43**(1), 1–54 (2015)
52. H. Freeman, Computer processing of line-drawing images. ACM Comput. Surv. **6**(1), 57–97 (1974)

53. E. Koh, J. Won, C. Bae, On-premise skin color modeling method for vision-based hand tracking, in *Proceedings 2009 IEEE 13th International Symposium on Consumer Electronics* (IEEE, Piscataway, 2009), pp. 908–909
54. Y. Zhao, W. Wang, Y. Wang, A real-time hand gesture recognition method, in *Proceedings 2011 International Conference on Electronics, Communications and Control* (IEEE, Piscataway, 2011), pp. 2475–2478
55. N. Bauer, P. Pathirana, P. Hodgson, Robust optical flow with combined Lucas-Kanade/Horn-Schunck and automatic neighborhood selection, in *2006 International Conference on Information and Automation* (IEEE, Piscataway, 2006), pp. 378–383
56. J. Shi, C. Tomasi, Good features to track, in *Proceedings 1994 Conference on Computer Vision and Pattern Recognition* (IEEE, Piscataway, 1994), pp. 593–600
57. M.S.M. Asaari, B.A. Rosdi, S.A. Suandi, Adaptive Kalman filter incorporated Eigenhand (AKFIE) for real-time hand tracking system. Multimed. Tools Appl. **74**(21), 9231–9257 (2015)
58. Y. Liu, P. Zhang, Hand gesture tracking using particle filter with multiple features, in *Proceedings International Symposium on Intelligent Information Systems and Applications* (2009), pp. 28–30
59. T. Ojala, M. Pietikainen, T. Maenpaa, Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Trans. Pattern Anal. Mach. Intell. **24**(7), 971–987 (2002)
60. N.D. Binh, E. Shuichi, T. Ejima, Real-time hand tracking and gesture recognition system, in *Proceedings of the GVIP* (2005), pp. 19–21
61. R. Shrivastava, A hidden Markov model based dynamic hand gesture recognition system using OpenCV, in *Proceedings 3rd International Advance Computing Conference* (IEEE, Piscataway, 2013), pp. 947–950
62. Z. Zivkovic, Improved adaptive Gaussian mixture model for background subtraction, in *Proceedings 17th International Conference on Pattern Recognition*, vol. 2 (IEEE, Piscataway, 2004), pp. 28–31
63. T. Coogan, G. Awad, J. Han, A. Sutherland, Real-time hand gesture recognition including hand segmentation and tracking, in *Proceedings Second International Conference on Advances in Visual Computing—Part I* (Springer, Berlin, 2006), pp. 495–504
64. E.-J. Holden, G. Lee, R. Owens, Automatic recognition of colloquial Australian sign language, in *Seventh IEEE Workshops on Application of Computer Vision, 2005. WACV/MOTIONS'05 Volume 1*, vol. 2 (IEEE, Piscataway, 2005), pp. 183–188
65. L. Goldschlager, A. Lister, *Computer Science: A Modern Introduction* (Prentice Hall, London, 1988)
66. S. Howard, T. Chowles, L. Reynolds, *Finger Talk: South African Sign Language (SASL) Dictionary* (Fulton School for the Deaf, Gillitts, 2011)
67. R. McKee, G. Kennedy, New Zealand sign language, in *Languages of New Zealand* (Victoria University Press, Wellington, 2005), pp. 271–297
68. W. Forman, The ABCs of New Zealand sign language: aerial spelling. J. Deaf Stud. Deaf Educ. **8**(1), 92–96 (2003)
69. J.A. Bickford, *The Signed Languages of Eastern Europe* (SIL International, Dallas, 2005)
70. Z. Ren, J. Yuan, J. Meng, Z. Zhang, Robust part-based hand gesture recognition using Kinect sensor. IEEE Trans. Multimed. **15**(5), 1110–1120 (2013)
71. A.I. Maqueda, C.R. del Blanco, F. Jaureguizar, N. García, Temporal pyramid matching of local binary subpatterns for hand-gesture recognition. IEEE Signal Process. Lett. **23**(8), 1037–1041 (2016)
72. P.S. Heckbert, A seed fill algorithm, in *Graphics Gems* (Academic Press Professional, New York, 1990), pp. 275–277
73. J. Sklansky, Finding the convex hull of a simple polygon. Pattern Recogn. Lett. **1**(2), 79–83 (1982)
74. N. Shimada, K. Kimura, Y. Shirai, Real-time 3D hand posture estimation based on 2D appearance retrieval using monocular camera, in *IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 2001. Proceedings* (IEEE, Piscataway, 2001), pp. 23–30

75. B. Shi, A.M. Del Rio, J. Keane, J. Michaux, D. Brentari, G. Shakhnarovich, K. Livescu, American sign language fingerspelling recognition in the wild, in *2018 IEEE Spoken Language Technology Workshop (SLT)* (IEEE, Piscataway, 2018), pp. 145–152
76. M. Elpeltagy, M. Abdelwahab, M.E. Hussein, A. Shoukry, A. Shoala, M. Galal, Multi-modality-based Arabic sign language recognition. IET Comput. Vis. **12**(7), 1031–1039 (2018)
77. R. Alzohairi, R. Alghonaim, W. Alshehri, S. Aloqeely, M. Alzaidan, O. Bchir, Image based Arabic sign language recognition system. Int. J. Adv. Comput. Sci. Appl. (IJACSA) **9**(3), 185–194 (2018)
78. B. Kang, K.-H. Tan, N. Jiang, H.-S. Tai, D. Tretter, T. Nguyen, Hand segmentation for hand-object interaction from depth map, in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (IEEE, Piscataway, 2017), pp. 259–263
79. S. Saremi, S. Mirjalili, A. Lewis, A.W.C. Liew, J.S. Dong, Enhanced multi-objective particle swarm optimisation for estimating hand postures. Knowl. Based Syst. **158**, 175–195 (2018)
80. Y. Cai, L. Ge, J. Cai, J. Yuan, Weakly-supervised 3d hand pose estimation from monocular RGB images, in *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 666–682
81. Y. Hu, Finger spelling recognition using depth information and support vector machine. Multimed. Tools Appl. **77**(21), 29043–29057 (2018)
82. I.L. Bastos, M.F. Angelo, A.C. Loula, Recognition of static gestures applied to Brazilian sign language (Libras), in *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images* (IEEE, Piscataway, 2015), pp. 305–312
83. H. Lahiani, M. Neji, Hand gesture recognition method based on HOG-LBP features for mobile devices. Procedia Comput. Sci. **126**, 254–263 (2018)
84. Z. Yang, Y. Li, W. Chen, Y. Zheng, Dynamic hand gesture recognition using hidden Markov models, in *Proceedings 7th International Conference on Computer Science and Education* (IEEE, Piscataway, 2012), pp. 360–365
85. T. Matsuo, Y. Shirai, N. Shimada, Construction of general HMMs from a few hand motions for sign language word recognition, in *MVA2013 IAPR International Conference on Machine Vision Applications* (2013), pp. 69–72
86. J. Hua, Z. Ju, D. Chen, D. Zhou, H. Zhao, D. Jiang, G. Li, Multiple features fusion system for motion recognition, in *International Conference on Intelligent Robotics and Applications* (Springer, Cham, 2019), pp. 445–455
87. P.V. Barros, N. Junior, J.M. Bisneto, B.J. Fernandes, B.L. Bezerra, S.M. Fernandes, Convexity local contour sequences for gesture recognition, in *Proceedings 28th Annual ACM Symposium on Applied Computing* (ACM, New York, 2013), pp. 34–39
88. W.-L. Lu, J.J. Little, Simultaneous tracking and action recognition using the PCA-HOG descriptor, in *Proceedings 3rd Canadian Conference on Computer and Robot Vision* (IEEE, Piscataway, 2006), pp. 1–8
89. C. Vogler, D. Metaxas, Handshapes and movements: multiple-channel American sign language recognition, in *International Gesture Workshop* (Springer, Berlin, 2003), pp. 247–258
90. W. Nel, An integrated sign language recognition system. Master's thesis, University of the Western Cape, Computer Science, 2014
91. M. Raptis, L. Sigal, Poselet key-framing: a model for human activity recognition, in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 2650–2657
92. J. Tilmanne, S. Hidot, T. Ravet, Mockey: motion capture as a tool for keyframing animation, QPSR Numediart Res. Program **2**(4) (2009) 119–124

Automatic Sign Language Manual Parameter Recognition (II): Comprehensive System Design

Mehrdad Ghaziasgar, Antoine Bagula, and Christopher Thron

1 Introduction

The automatic sign language recognition system described in this chapter is an overview of the design that is extensively delineated in [1]. Reference [1] provides more in-depth details concerning the design, parameter optimisation, and experimental verification of the system.

The system consists of two main components: hand retrieval and manual parameter representation and recognition. These two components are described separately in the two following sections of this chapter.

2 Hand Retrieval

This section describes the hand retrieval component of the proposed framework. The hand retrieval component aims to find and track the hands in a video sequence. Figure 1 provides an overview of the hand retrieval component. The "Start" sub-component in the figure specifies the starting point of processing the first time that the system runs. Subsequent steps in the process are:

M. Ghaziasgar (✉) · A. Bagula
Department of Computer Science, University of the Western Cape, Cape Town, South Africa
e-mail: mghaziasgar@uwc.ac.za

C. Thron
Department of Science and Mathematics, Texas A&M University-Central Texas, Killeen, TX, USA

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_4

- A sequence of frames is captured by a monocular view.
- Before tracking is initiated, the user holds up both hands with open palms at the top-right of the figure. The hand detection sub-component locates the open palms of the user. These locations are used to enhance skin detection and initialise the hand tracking sub-component.
- The skin detection and motion detection sub-components are invoked on the input frame in parallel separate threads. Once both threads have completed processing, the skin and motion images that result from these sub-components are combined using a logical AND operation in order to limit sources of stationary skin-coloured noise.
- The resulting image—henceforth referred to as the "moving skin" image—is used as input to the hand tracking sub-component, which tracks both hands.

With the positions and sizes of the hands determined, processing continues to the manual parameter representation and recognition component explained in the next section.

The sub-components in Fig. 1 are very high-level abstractions that represent a workflow of tasks. The following subsections expand on each of these sub-components, following the progression through Fig. 1. In each case, the workflow pertaining to the relevant sub-component is set out and explained in detail.

2.1 Input Capture

Capturing frames in real-time from a camera is computationally costly and is typically the biggest impediment to processing in real-time for many computer vision applications [2], as the rate of input capture by the camera is typically very different from the processing speed of the application. Because of this, a general lack of synchronisation develops between the camera and the processing procedure of the application. Figure 2 shows a proposed solution to this problem, which is to create a separate thread (the camera reading thread) that continuously reads a frame from the camera as soon as one is available and stores it into a shared buffer that is available to the main processing thread. If the application is occupied with processing and a new frame is acquired, the frame is placed in the buffer for processing. When processing returns to the beginning of the cycle, the new frame is retrieved from the buffer and processing continues with virtually no delay. If multiple frames are made available by the camera while the application is engaged in processing, the most recent frame overwrites the currently available frame in the buffer: only one frame is available in the buffer at any time. This prevents the application from having to handle backlogs that may accumulate.

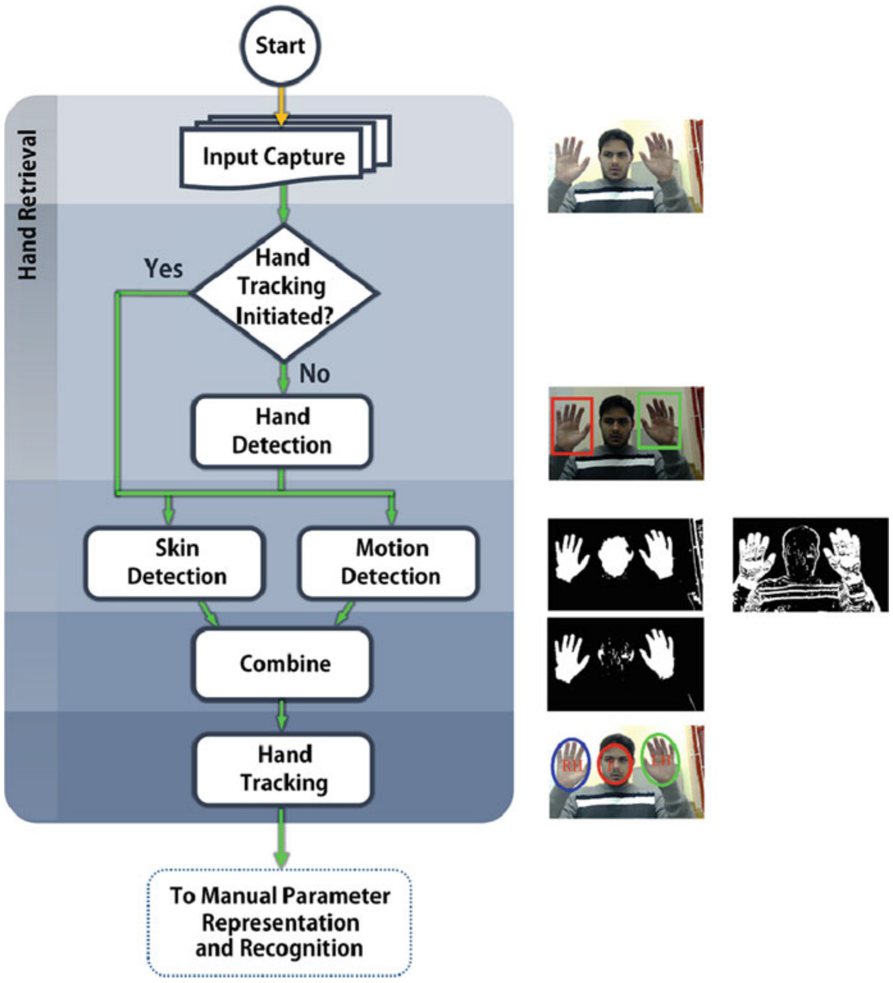


Fig. 1 Overview of the hand retrieval component of the proposed sign language manual parameter recognition and transcription framework

2.2 *Hand Detection*

Figure 3 provides a graphical summary of the steps involved in detected the hands. Referring to the figure, before hand tracking has been initialised, the user is asked to hold up his/her hands with open palms similar to those shown in the input frame at the top-right of Fig. 1. The input image is converted to its greyscale equivalent, followed by the use of adaptive Gaussian thresholding and an image inversion to produce a binary variant of the greyscale image in which prominent contours have been highlighted [3]. Cross-correlation template matching [4] is then used to find the

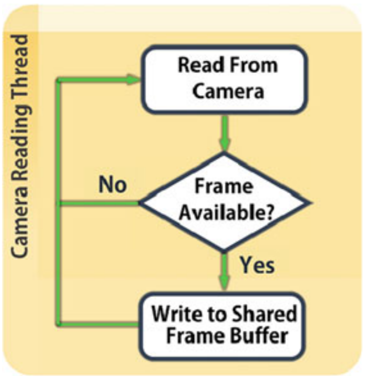


Fig. 2 Modified input capture component: a camera reading thread continuously reads a frame from the camera as soon as one is acquired and places it in a buffer that is shared with the main thread. The main thread is able to retrieve this frame and commence processing

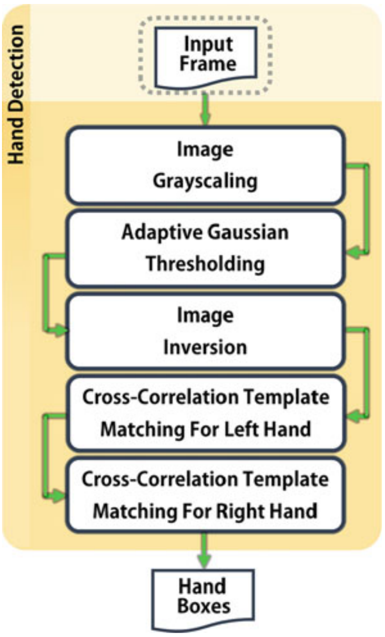

Hand Detection
Input Frame
Image Grayscaling
Adaptive Gaussian Thresholding
Image Inversion
Cross-Correlation Template Matching For Left Hand
Cross-Correlation Template Matching For Right Hand
Hand Boxes

Fig. 3 Overview of the hand detection procedure

position of maximum correlation between this binary contour image and silhouette templates of the left and right hands with open palms. The contour templates are depicted in Fig. 4. Cross-correlation template matching is known to be scale-sensitive [4], so the user is required to sit at a specific distance from the camera. In addition, a hierarchical approach [5] is used to provide a measure of scale invariance. This involves making two smaller-resolution copies of each original template and applying these to the search image, in addition to the original template. The final correlation for each hand is taken as the maximum of the correlations of the three relevant templates. An empirically determined correlation threshold is then used

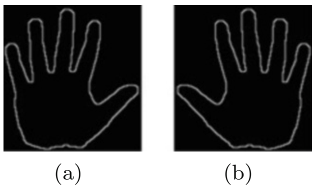


Fig. 4 Contour hand templates: (**a**) right hand; (**b**) left hand

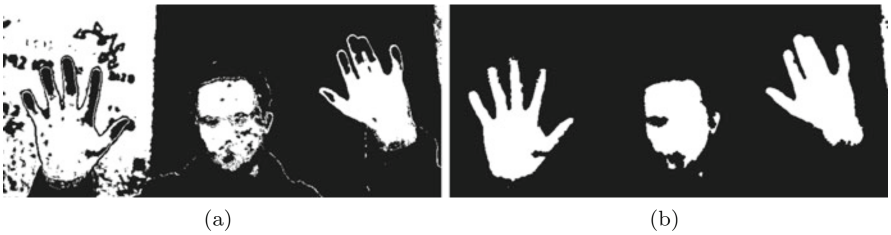


Fig. 5 Comparison between the original adaptive histogram-based skin detection and the proposed enhanced skin detection: (**a**) adaptive histogram-based skin detection result; (**b**) result obtained from the enhanced skin detection procedure

to determine whether the correlation maximum represents a true or false positive detection.

This procedure results in two boxes, one enclosing the left hand and one enclosing the right hand, in the input image. The boxes are characterised by the (x, y) coordinates of their top left and bottom right corners. These boxes are used in the enhanced skin detection sub-component before hand tracking has been initialised. They are also used to initialise hand tracking.

2.3 *Skin Detection*

With the two hand boxes obtained, skin detection is applied. For this purpose, an enhanced skin detection (ESD) approach was developed, which gave significant improvements over adaptive histogram-based skin detection (see Fig. 5). ESD uses the face histogram as a starting point for detecting the skin of the hands and incorporates additional skin colour information from the hands themselves into the skin histogram. This is achieved by employing contextual information—the sizes and locations of the hands—from a hand detection or hand tracking procedure in the skin detection procedure. The resulting skin model is significantly more representative of the specific skin colour of the hands, in addition to the face.

Figure 6a provides an overview of the proposed ESD approach. The inputs to ESD are the original RGB input frame from the camera and the hand boxes as input. Using these inputs the face is detected and located, and a histogram of a portion of the face is computed. Using the face histogram as an initial skin model, coupled

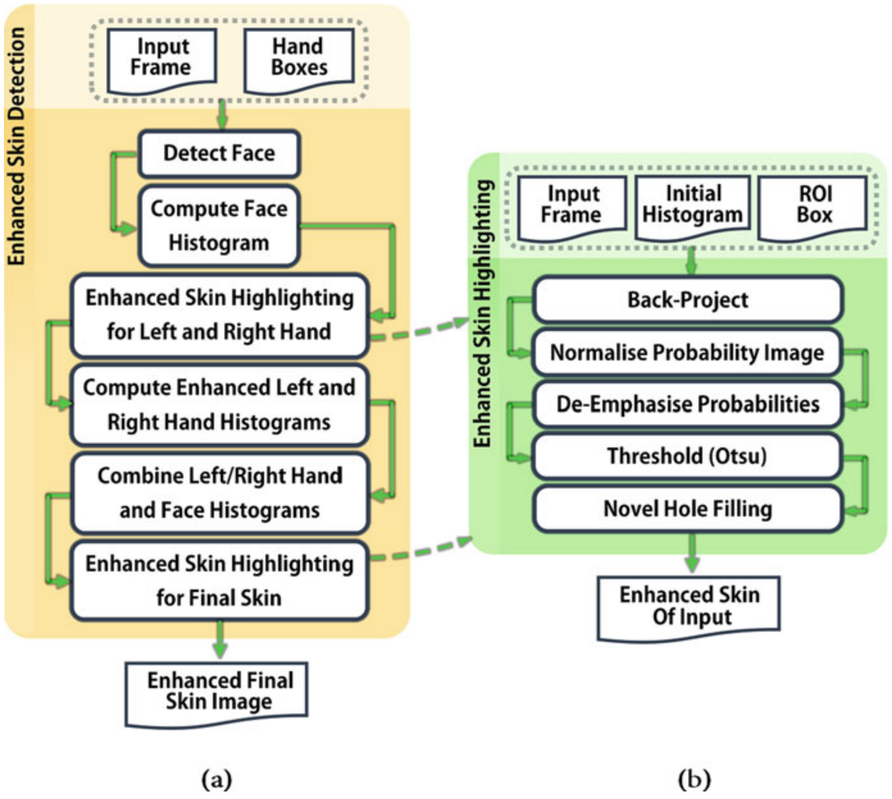

Enhanced Skin Detection
Input Frame
Hand Boxes
Detect Face
Compute Face Histogram
Enhanced Skin Highlighting for Left and Right Hand
Compute Enhanced Left and Right Hand Histograms
Combine Left/Right Hand and Face Histograms
Enhanced Skin Highlighting for Final Skin
Enhanced Final Skin Image
(a)
Enhanced Skin Highlighting
Input Frame
Initial Histogram
ROI Box
Back-Project
Normalise Probability Image
De-Emphasise Probabilities
Threshold (Otsu)
Novel Hole Filling
Enhanced Skin Of Input
(b)

Fig. 6 Overview of the proposed skin detection sub-component: (**a**) overview of the proposed enhanced skin detection procedure; (**b**) overview of the enhanced skin highlighting procedure which is a crucial part of the overall enhanced skin detection procedure

with the hand boxes, an enhanced skin highlighting (ESH) procedure is called up to obtain more accurate images of the skin in the hand regions than would have been possible using adaptive histogram-based skin detection. Steps within the ESH submodule are shown in Fig. 6b: these steps will be explained in more detail in Sect. 2.6.

The enhanced skin images of the hands obtained from the ESH procedure are used to obtain additional skin-coloured regions of the hands in the input frame that were not represented in the face histogram. Computing histograms of these hand regions yields colour models that are specific to, and more representative of, each hand. These histograms are incorporated into the face histogram by means of a summing operation. The resulting combined histogram is an enhanced model of the hands' skin colour. Using this enhanced histogram, the ESH procedure is called up again, now to obtain a final enhanced skin image on the entire original input frame.

The different steps shown in Fig. 6 are described in further detail in the subsections below.

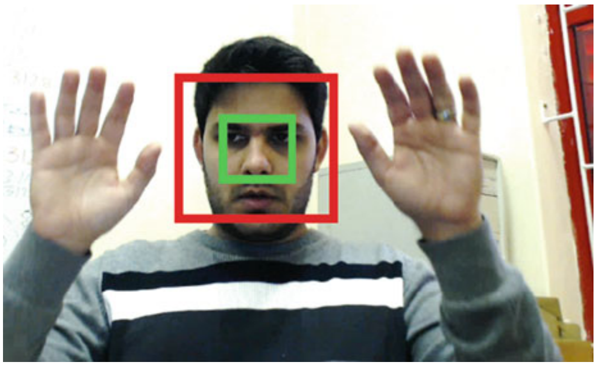


Fig. 7 Face detected box (red) and face sub-region used to compute a histogram (green)

2.4 Face Detection

The Viola–Jones face detection framework [6] is used to detect and locate the user's face. Viola–Jones is a very efficient and accurate face detection approach and is perhaps the most prominent face detection approach in the literature [7]. Figure 7 depicts a red box representing a detected face by the algorithm. The face is thus located in the overall proposed skin detection sub-component.

2.5 Face Histogram Computation

Once the position and size of the face have been determined, a histogram of a portion of the face is computed, which serves as the initial skin model. The region on which the histogram is computed is in the centre of the facial box, and half its width and height. This is visually illustrated by the green box in Fig. 7. The histogram computed is a 2-dimensional histogram computed on the hue (H) and saturation (S) channels of the HSV colour space. The conversion of the default RGB input frame to the HSV colour space is achieved by means of a nonlinear transformation [8]. It was experimentally determined that 12 hue bins and 12 saturation bins provided the best tradeoff between precision and recall rate out of a large range of possible combinations.

2.6 Enhanced Skin Highlighting Principle and Its Application to the Left and Right Hands

After the face histogram is obtained, the ESH procedure is applied twice, once for each hand, to obtain more accurate skin images of the hands. The ESH procedure is an enhanced form of the original adaptive histogram-based skin highlighting.

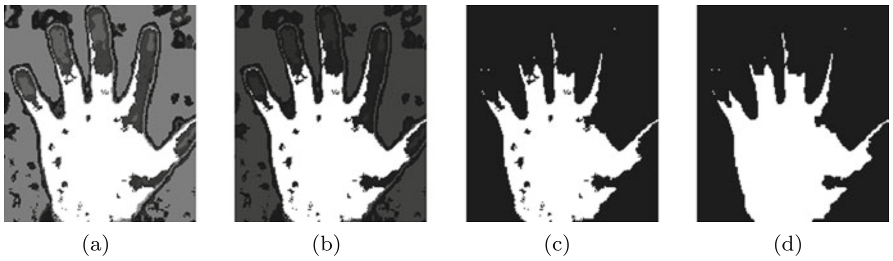


Fig. 8 Figures (**a**)–(**d**) show respectively: face histogram back-projected onto the hand; de-emphasised probability image; thresholded image; holes filled in

Fig. 9 Overview of the proposed novel hole-filling procedure

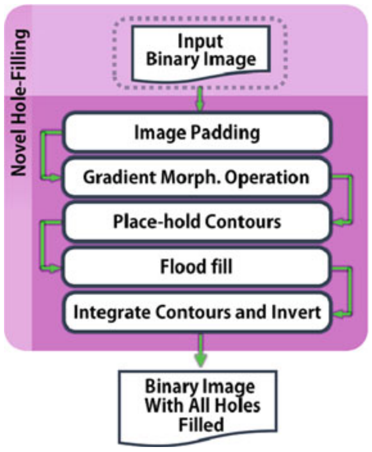

Novel Hole-Filling
Input Binary Image
Image Padding
Gradient Morph. Operation
Place-hold Contours
Flood fill
Integrate Contours and Invert
Binary Image With All Holes Filled

Figure 8a–d illustrates the steps of the ESH procedure, which are listed in Fig. 6b and described as follows.

First, given the input image, the initial face histogram of the user's face and the boxes enclosing the left and right hands, ESH back-projects the face histogram onto the hand region resulting in the skin probability images; Fig. 8a shows an example result for a right hand image. Nonlinear rescaling is applied to the probabilities so that all but the most probable skin pixels are de-emphasised. Figure 8b shows how this step effectively reduces background noise. Next, Otsu thresholding [9] is applied. Otsu thresholding calculates a dynamic threshold value that divides two desired pixel classes—skin and non-skin in this case—such that the weighted intra-class variance is minimised. As shown in Fig. 8c, this eliminates almost all non-skin pixels, but also a large proportion of skin pixels. Finally, a novel hole-filling (NHF) procedure is applied to fill in the holes in hand-coloured regions that are not represented in the face histogram, but can help more effectively highlight the hands. The NHF procedure is a multi-step procedure depicted in Fig. 9 that takes in a binary input image and closes up holes without any loss of information. A hole, in this case, is a black background cluster that is completely surrounded by skin pixels,

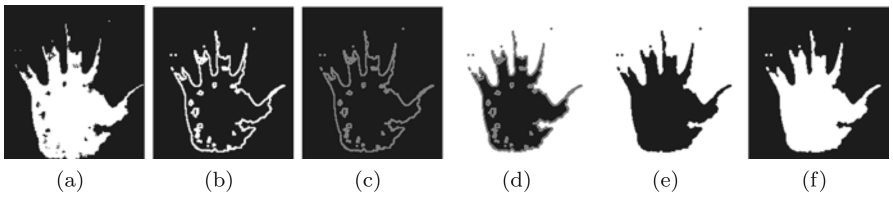


Fig. 10 The novel hole-filling procedure: Figures (**a**)–(**f**) are as follows (left to right): hand image padded with a 10-pixel-thick border; application of the gradient morphological operation; conversion of white pixels to grey; application of a flood fill operation at (0, 0) of the image; conversion of grey pixels to black; inversion of the image for final filled skin

or one that has a very tight opening of 3 pixels or less. The application of each step of the NHF is visually illustrated in Fig. 10. The holes filled in by NHF represent crucial skin-coloured regions that are unrepresented in the face histogram. Further details of the steps shown in the figure may be found in [1].

2.7 Computation of Enhanced Histograms for the Hands and Integration into the Face Histogram

Given the skin images of each hand, which have a greater proportion of skin-coloured regions highlighted, histograms of the hands in the regions of the input frame corresponding to skin in the hand skin images are computed. These histograms provide representation of hand-coloured regions that were not present in the face histogram and can help highlight a greater portion of the hands in a subsequent histogram back-projection.

At this point, the face, left hand and right hand histograms are combined by a summing operation to obtain a final enhanced skin histogram.

2.8 Enhanced Skin Highlighting for the Final Skin Image

Given the final enhanced skin histogram, the final step in the proposed skin detection procedure is to re-apply the ESH procedure to the entire input frame with the enhanced histogram. Figure 11 shows an example of the results obtained by the proposed skin detection, compared to adaptive histogram-based skin detection. The proposed procedure yields vastly improved performance.

Experimental verification of the accuracy of the ESD method is provided in [1].

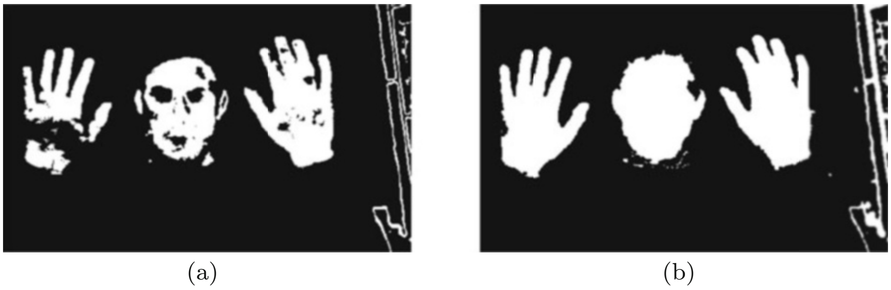


Fig. 11 Comparison between the proposed enhanced skin detection and adaptive histogram-based skin detection: (**a**) skin image of the original adaptive histogram-based skin detection; (**b**) skin image of the enhanced skin detection method

2.9 *Motion Detection*

In parallel with the computation of the skin image, a motion detection procedure is used to highlight motion in the input frame.

The motion detection method used was originally proposed by Stuaffer and Grimson [10] and was later modified by Zivkovic [11]. The method models the appearance history of each pixel as a mixture of Gaussians in the current frame and uses an online approximation to update the model of each pixel in each frame over a specific history length. This includes the number of Gaussians and their respective parameters that most effectively model the pixel in the frame. Each pixel is then assigned to either the stationary background or moving foreground, depending on whether or not the Gaussian distribution that most effectively models the pixel is deemed to be part of the background.

A formal description of this algorithm is provided in [1].

2.10 *Combination of Skin and Motion Images*

With the enhanced skin image and the motion image obtained, a logical AND operation is carried out to combine the images. For convenience and comparison, the skin and motion images are provided in Fig. 12a and b and are combined in Fig. 12c.

A large section of skin-coloured noise can be observed on the right side of the skin image in Fig. 12a. Similarly, the motion image in Fig. 12b contains many highlighted regions that are not skin, such as the clothing areas on the arms and chest of the user.

Many sources of noise such as these are effectively eliminated in the combined image in Fig. 12c, which highlights moving skin. A final application of the NHF procedure on this image closes up holes as seen in Fig. 12d. The combined image

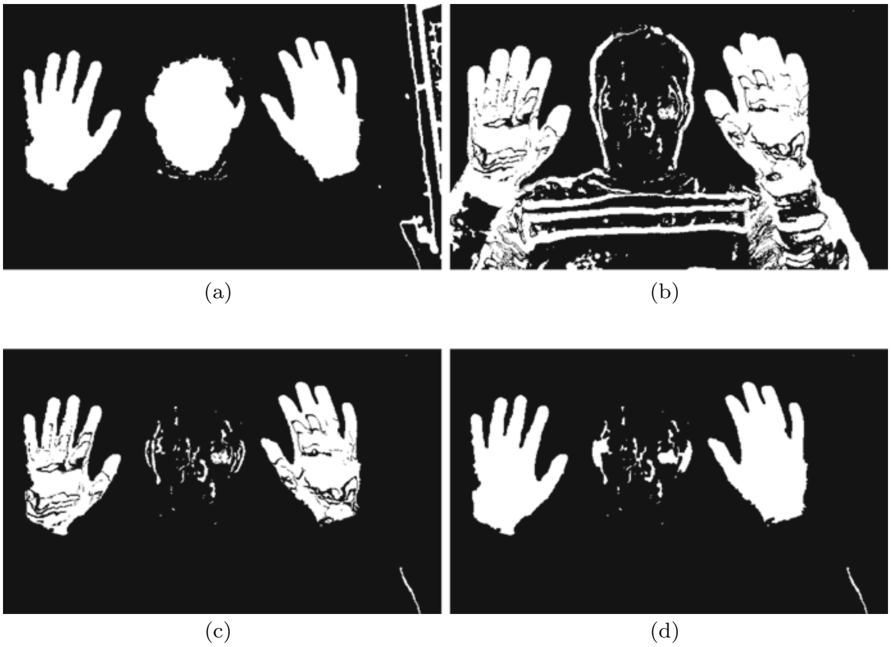


Fig. 12 Combining the skin and motion image: (**a**) the skin image; (**b**) the motion image; (**c**) the combined image; (**d**) the combined image with holes filled in—“moving skin image”

with holes filled in is henceforth referred to as the “moving skin image”. This image is used in the hand tracking sub-component in the next section.

2.11 Hand Tracking

Figure 13 is an overview of the hand tracking sub-component. The procedure takes as inputs the moving skin image, the skin image, and the face and hand boxes, all of which have been obtained in previous steps. If hand tracking has not yet been initiated, a tracking initialisation procedure is invoked to locate the hands and face using the hand and face boxes from previous steps, as well as to locate all other skin-coloured objects. If tracking was initiated in a previous processing cycle, then a tracking update (the core of the tracking sub-component) is invoked to associate the currently observed skin clusters in the moving skin image with the known tracked objects in previous frames, thereby adapting the tracking state to the current observations, as will be explained in detail.

Section 2.11.1 below first describes the data association concept towards object tracking. The implementation of the concept is then proposed and described in detail in Sects. 2.11.2 and 2.11.3 which describe, respectively, the tracking initialisation

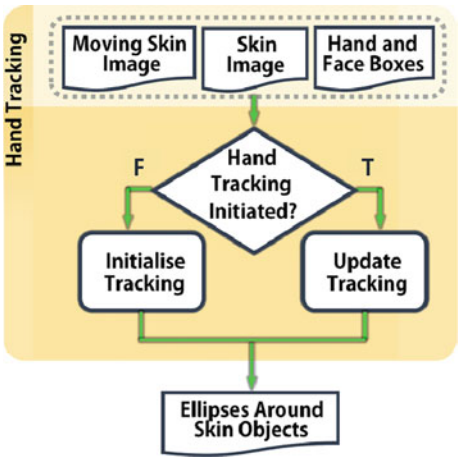


Fig. 13 Overview of the hand tracking sub-component

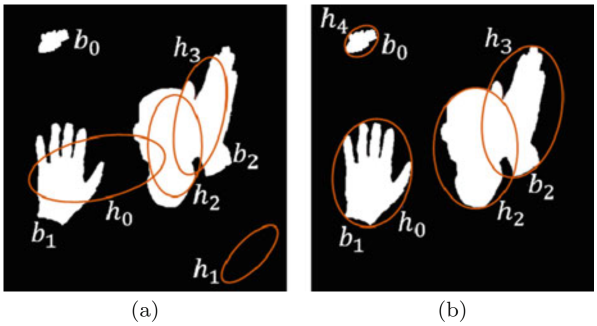


Fig. 14 Resolution of hypotheses on a set of observed blobs: (**a**) the blobs observed and the current state of hypotheses in the current frame before resolving and updating them; (**b**) updated hypotheses after the association and resolution rules have been applied

and tracking update steps of the sub-component. A worked example of the tracking update step is given in [1].

2.11.1 Data Association for Object Tracking

In general terms, data association [12] applied to object tracking is applied to a binary image in which objects to be tracked are represented by highlighted clusters/blobs, and everything else in the scene is represented as black background. Figure 14 illustrates the data association process. The images show "blobs" (b_0, b_1, b_2) and hypotheses (h_0, h_1, h_2, h_3): each hypothesis h_j is parametrically described by an ellipse with a centroid $C_j(c_{x_j}, c_{y_j})$, respective major and minor axes α_j and β_j, and an angle of inclination in the 2D plane of the image θ_j. h_j is therefore fully described as $h_j(C_j, \alpha_j, \beta_j, \theta_j)$. It is apparent that the hypotheses in Fig. 14 do not accurately represent the blobs observed in the current image, due to changes in location, shape, and/or orientation of the blobs from the previous to the current frame. Thus, a tracking update amounts to arriving at the hypotheses in Fig. 14b from Fig. 14a, which in turn amounts to determining the associations between the current hypotheses and observed blobs.

Observing Fig. 14a in more detail, it is observed that: blob b_0 does not have a hypothesis representing it, since it has possibly just entered the frame; blob b_1 is represented by one hypothesis h_0 which also represents a portion of blob b_2, possibly attributed to an overlapping between blobs b_1 and b_2 in the previous frame; and b_2 has two additional hypotheses h_2 and h_3 assigned to specific portions of the blob. Hypothesis h_1 does not represent any blobs, possibly because the blob that it represented in one or more previous frames has moved out of the camera view and is no longer visible in the frame.

The association between a hypothesis h_j and blob b_i, and the extent of this association, is established by means of a distance function D which provides the distance of each pixel $p(p_x, p_y)$ in b_i from h_j given by:

$$D(h_j, p) = \sqrt{\bar{z} \cdot \bar{z}} \tag{1}$$

where

$$\bar{z} = \begin{bmatrix} \cos(\theta_j) & -\sin(\theta_j) \\ \sin(\theta_j) & \cos(\theta_j) \end{bmatrix} \begin{bmatrix} \frac{p_x - c_{x_j}}{\alpha_j} \\ \\ \frac{p_y - c_{y_j}}{\beta_j} \end{bmatrix} \tag{2}$$

Depending on whether point p is inside, on the boundary of, or outside ellipse h_j, D provides a value that is less than, equal to, or greater than 1. Therefore, if $D(h_j, p) \leq 1$, the following deductions can be made: (1) point p is in or on hypothesis h_j; (2) h_j predicts and tracks at least a portion of blob b_i; and (3) the hypothesis h_j should still exist, i.e., its existence is still supported. Therefore, blobs b_1 and b_2 in Fig. 14a support the existence of hypothesis h_0 which predicts b_1 and a smaller portion of b_2. On the other hand, no points of any of the blobs support the existence of h_1, which does not predict or track any part of any of the blobs. This hypothesis should therefore be eliminated.

On the other hand, if $D(h_j, p) > 1$ for all hypotheses $\{h_j | (H-1) \geq j \geq 0\}$ and all points p in a specific b_i, it can be deduced that blob b_i is completely unrepresented and not predicted or tracked by any of the currently existing hypotheses. b_0 in Fig. 14a is an example of this case. Blobs of this type are each assigned a new hypothesis by fitting an ellipse onto the points of the blob. The fitted ellipse is taken as the covariance ellipse of the convex hull of the blob. Figure 15a illustrates the points of the convex hull of the blob represented by a series of red dots. Figures (b) and (c) demonstrate the superiority of using the blob's

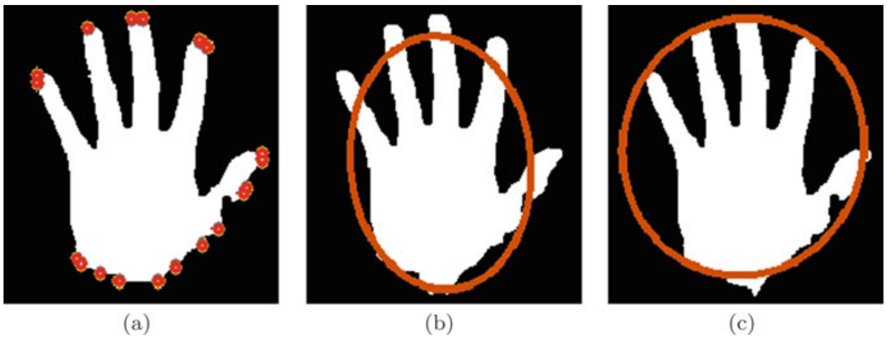


Fig. 15 Improved ellipse-fitting: (**a**) Convex hull of the blob represented by points in red; (**b**) covariance ellipse of blob; (**c**) covariance ellipse of convex hull (proposed ellipse-fitting method)

convex hull for the ellipse calculation, rather than the entire blob. Obtaining an ellipse that effectively covers the entire blob is crucial to segmenting the entire hand without omission in the hand segmentation step ahead of hand shape and hand orientation recognition described in Sect. 3.

It is in this way that any new or "lost" blobs observed in any current frame are detected and assigned a new tracking hypothesis. Given the set of blobs and hypotheses determined as above, an algorithm has been developed to assign each pixel in all blobs to a unique hypothesis (for a detailed description of the algorithm, see [1]). The application of this algorithm to Fig. 14a results in the hypothesis arrangement of Fig. 14b. In this case, hypotheses h_0, h_2, and h_3 have been repositioned, a new hypothesis h_4 has been created, and h_1 has been eliminated. Finally, to enable identification of hypotheses in the next frame, a prediction of the position of the centroid of each hypothesis is obtained by linear extrapolation.

2.11.2 Tracking Initialisation

The tracking initialisation procedure ensures that hypotheses are assigned to the three principal regions: left hand, right hand, and face. Several steps are carried out during initialization to ensure that all three principal regions are captured, and stray blobs are not misidentified. The two hands and face are considered to be correctly identified if blobs are found whose centroids lie in each of the three bounding boxes described in Sects. 2.2 and 2.3. If initialisation happens to fail during the first frame, the initialisation procedure is applied to subsequent frames until a successful result is obtained. A detailed description of the initialisation procedure may be found in [1].

2.11.3 Tracking Update

Once initialisation has successfully identified and assigned hypotheses to the face and hands, and other skin-coloured objects in the moving skin image, a tracking update is carried out on every subsequent frame to track these objects.

The moving skin image, henceforth referred to as "the image" for convenience, is computed on the input frame as before. During the process, the face box is also retrieved. Given a list of hypotheses that represented objects in the previous frame, an extended algorithm is applied that is capable of resolving any number of hypotheses and blobs in any configuration. The steps of this procedure, as well as a specific worked-out example of tracking update, may be found in [1].

Once the hands have been located and tracked successfully, processing can proceed to isolate the hands, extract relevant salient features from the hands, and recognise and represent the manual sign language parameters. All this is described in the next section.

3 Manual Parameter Representation and Recognition

This section describes the second component of the proposed framework—the manual parameter representation and recognition (MPRR) component. Given that there are now ellipses around each hand from the hand retrieval component, the aim of this component is to accurately isolate the hands, represent the hands in terms of relevant salient features, recognise the relevant manual parameters, and represent them in SignWriting Markup Language (SWML).

Figure 16 shows the MPRR component within the context of the entire proposed framework. The initial sub-component within MPRR is labelled "hold detection". This sub-component detects movements and pauses, referred to as "holds", of each hand. More precisely, a hand is assumed to be in a "hold" state—or simply hold for short—if it stops moving or changes direction. If neither hand is in a hold, processing returns to the input capture sub-component of the hand retrieval component. Only when a hold is detected on either hand does processing proceed to accurately segment the relevant hand, represent it in terms of normalised salient features to obtain a feature vector, carry out hand orientation and hand shape recognition on that hand, and finally produce an SWML transcription thereof. If both hands are in a hold, this procedure is carried out for both hands.

It is observed in the figure that recognition is a two-stage process. First, a hand orientation classifier predicts the orientation of each hand that is in a hold, regardless of hand shape. Thereafter, depending on the predicted hand orientation of the relevant hand, one of a set of hand shape classifiers that is specifically trained to recognise hand shapes in the hand orientation predicted is invoked to determine the hand shape of the relevant hand. Once again, note that this two-stage procedure is invoked separately for each hand if both hands are in a hold.

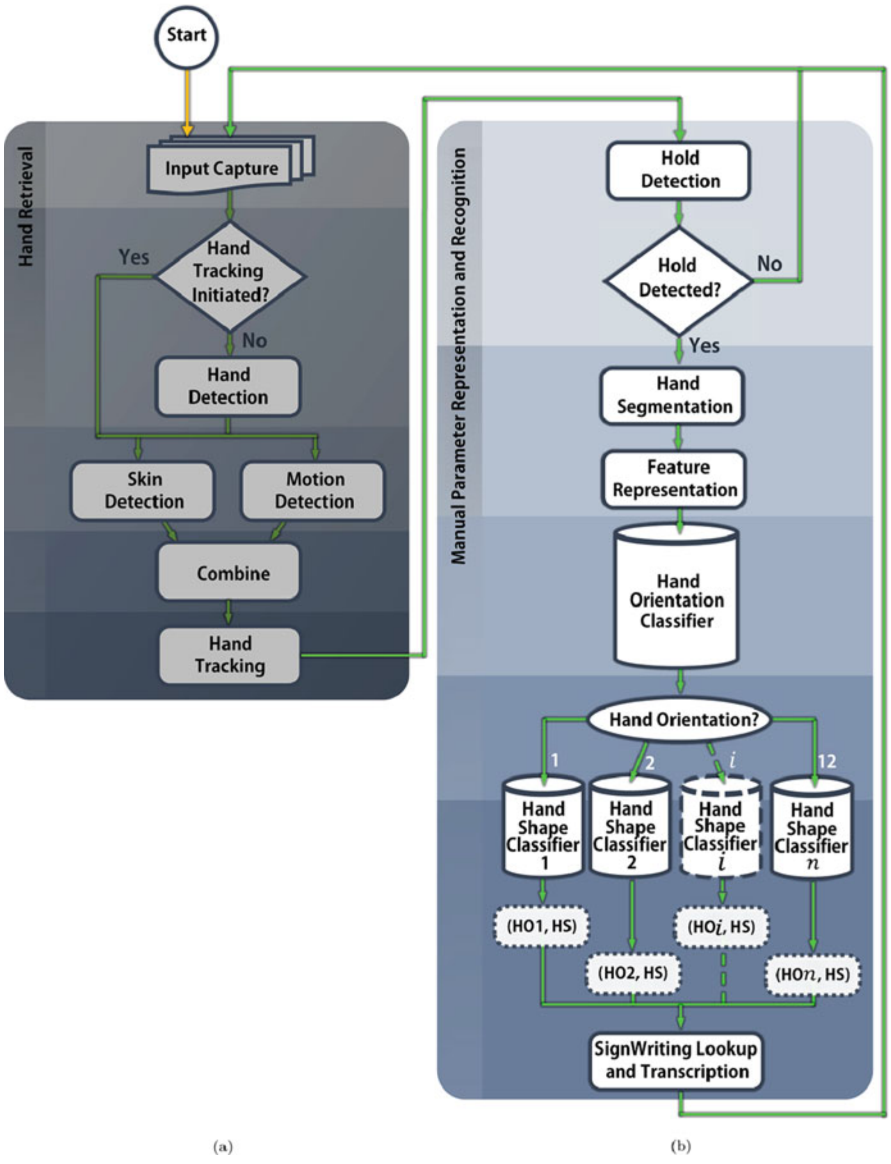


Fig. 16 Overview of the combined sign language manual parameter recognition and transcription framework: (**a**) the hand retrieval component; (**b**) the manual parameter representation and recognition component

At this stage, all four sign language manual parameters have been obtained. They are then represented using the corresponding SWML notation using a relatively simple symbol lookup and representation. Once this representation is complete, processing returns to retrieving a new frame in the hand retrieval component.

The following subsections describe the following sub-components of the MPRR component in greater detail: hold detection; hand segmentation; feature representation; hand orientation and hand shape recognition; and SignWriting lookup and transcription in SWML format.

3.1 Hold Detection for Motion Representation

The hold detection methodology is based on the observation that sign language gestures may be taken as fully formed when the relevant hand stops or changes direction [13]. A key-framing approach is used to select time steps where holds occur, thus providing a highly compact representation that can fully convey changes in any sequence [14, 15]. The use of key-framing leads to large savings in processing time, as well as increased accuracy since hands that are in motion are usually transitioning between gestures and cannot be definitively classified.

This research proposes the use of two finite state machines (FSM) depicted in Fig. 17 to model movements and holds for the two hands. Referring to the figure, the FSM defaults to a *hold* state. If the relevant hand remains within a specific radius of the current hold location, a transition to the *holding* state is invoked. The FSM remains in this state as long as the relevant hand remains within a specific radius of the current hold location. If the relevant hand moves more than a specific distance away from the current hold location, a transition to the *movement* state is invoked. The FSM remains in this state as long as the hand continues moving and the hand

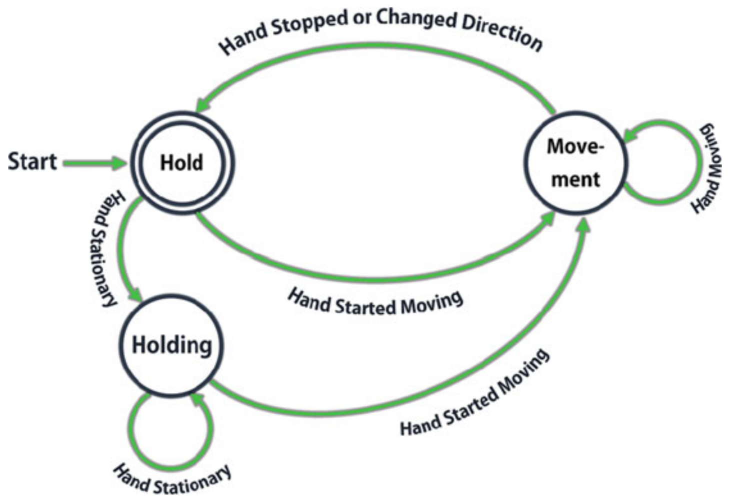


Fig. 17 The finite state machine used to model movements and holds of the hands

does not change its direction of motion. If the hand stops or changes its direction, a transition to the *hold* state is invoked, and the cycle is repeated. It is observed that the *hold* state is an accept state since further processing is invoked every time the FSM transitions to this state. Also, the reason for maintaining two similar states *hold* and *holding*, rather than only a single *hold* state, is to ensure that processing on any one hold takes place only once, on the *hold* state. Any subsequent frames that maintain the same hold are absorbed by the *holding* state, without invoking further processing.

3.1.1 Determining When the Hand Starts Moving

While in the *hold* or *holding* state, a check is carried out on every frame to determine if the hand has started to move. This is achieved by determining if the current ellipse centroid for the hand has moved more than a specific distance away from the hold location. Assuming that the hand has a centroid $C = (C_x, C_y)$ and the current hold location is $H = (H_x, H_y)$, a thresholding function $S_M(C, H)$ determines if the hand has started to move as follows:

$$S_M(C, H) = \begin{cases} \text{Moving} & \text{if } \sqrt{(H_x - C_x)^2 + (H_y - C_y)^2} \geq t_m \\ \text{Stationary} & \text{if } \sqrt{(H_x - C_x)^2 + (H_y - C_y)^2} < t_m \end{cases} \quad (3)$$

where t_m is an empirically determined threshold that determines how far the hand can be displaced before it is considered to be moving.

3.1.2 Determining Stops or Changes in Direction of the Hand

While in the *movement* state, a check is carried out on every frame to determine if the hand has stopped or changed direction. A dynamic method of achieving both checks simultaneously is to use estimates of hand velocities in both x- and y-directions. Changes in hand velocities can be estimated by using hand centroid locations from several consecutive frames, as shown in Fig. 18. In Fig. 18c, e we see plots of the x and y components of the right hand's centroid for seven consecutive frames, for the hand-raising gesture shown in Fig. 18a. In each pane, two lines are fitted: one line uses the first four points, and the other uses the last four points. The two slope estimates in each pane agree, indicating that no hold has occurred within the recording period. On the other hand, in panes Fig. 18d, f we see the corresponding x and y component plots for the raise-and-hold gesture shown in pane Fig. 18b, together with line fits for the first four and last four points as in Fig. 18c, e. In this case, the two x slopes in Fig. 18d agree, but the two y slopes in Fig. 18f do not: there is an "elbow" in the y-slope plot, which indicates that a hold has occurred at the fourth point. The hold detection sub-component performs this calculation for

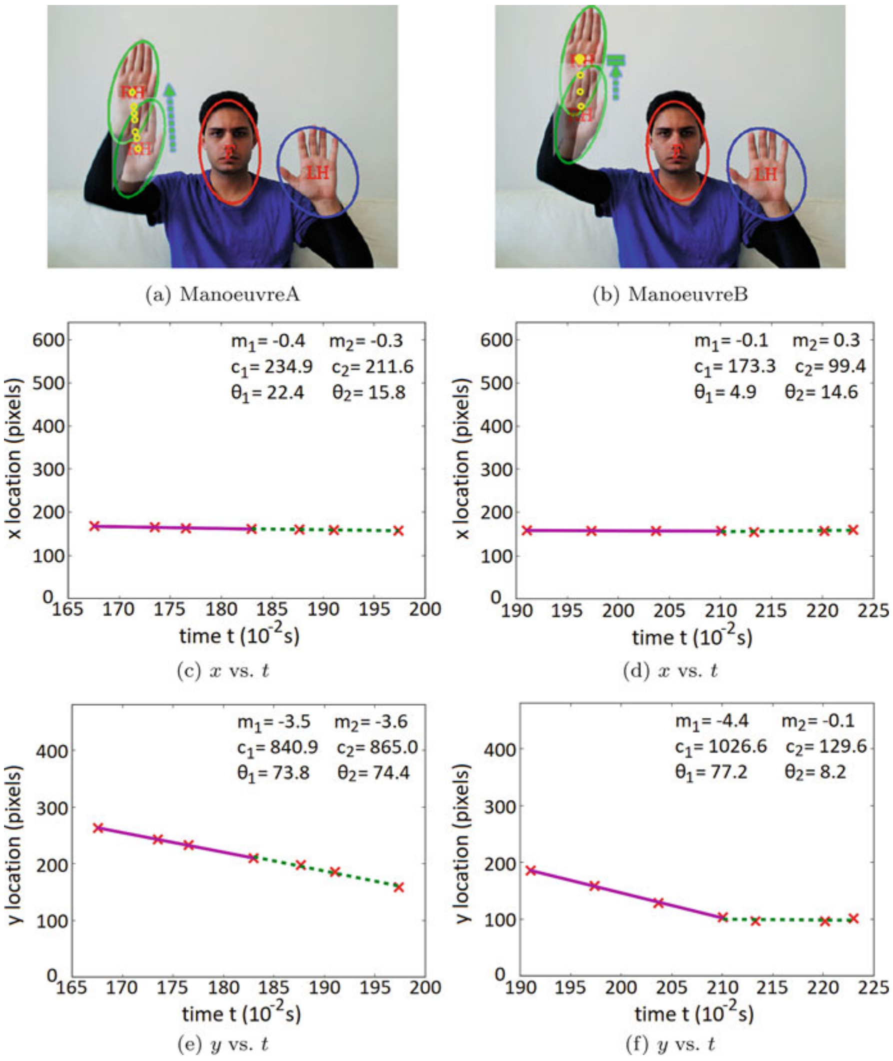


Fig. 18 Tracking and plots of the last seven x and y locations of the right hand versus time t in two example hand manoeuvres: (left column) a steady upward motion of the hand at time $t = 197 \times 10^{-2}$ s; (right column) the hand slowing down to a stop at time $t = 224 \times 10^{-2}$ s: (top row) illustration of tracking showing the hand centroids (yellow points) of the last seven time steps; (middle row) plot of x location of the right hand versus t for the last seven time steps; (bottom row) plot of y location of the right hand versus t for the same time steps. (**a**) Manoeuvre A. (**b**) Manoeuvre B. (**c**) x vs. t. (**d**) x vs. t. (**e**) y vs. t. (**f**) y vs. t

every frame, and if the difference in angles between the two lines for either x or y component exceeds an (empirically determined) threshold, then a hold is indicated.

The reader is referred to [1] for further details on this procedure.

4 Hand Segmentation

Given a hand that has entered the *hold* state, the goal of hand segmentation is to isolate only relevant portions of the input frame pertaining to that hand. The result of hand segmentation is, ideally, a tightly fit image containing only pixels of the relevant hand. An overview of the proposed hand segmentation procedure is provided in Fig. 19.

The procedure uses the greyscale equivalent of the original input frame, the skin image and the ellipse hypothesis of the hand to be segmented, all obtained/produced in previous sub-components of the framework. An overall explanation of this procedure following the progression through Fig. 19 follows.

A non-oriented bounding box, i.e., aligned with the x- and y-axes, of the hand hypothesis is computed and isolated from the greyscale image. Regions of the resulting image that are skin regions falling inside the hand hypothesis are isolated, and non-skin or outside regions eliminated, by means of a logical **AND** operation between this image and the corresponding sub-region of the skin image, using an image of the hand hypothesis as a mask. The moving skin image cannot be used here since the hands "disappear" in the image when the hands are stationary. Given hand tracking and skin detection sub-components that are sufficiently accurate and comprehensive, the resulting image should contain a single connected object representing the hand, with some small clusters of isolated skin-coloured noise. It is generally sufficient at this stage to simply eliminate the small noise clusters by eliminating all but the largest connected object.

However, to provide assurance for borderline cases in which the skin detection sub-component produces the hand as two or more disjoint connected skin clusters

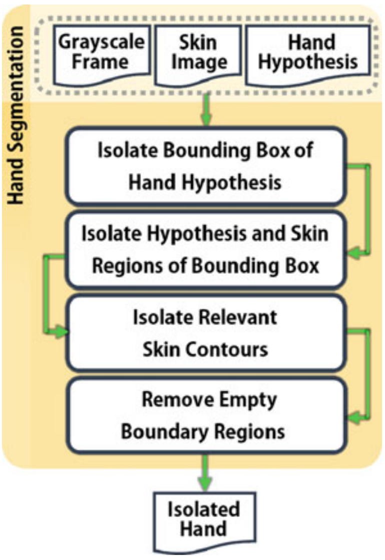

Hand Segmentation
Grayscale Frame
Skin Image
Hand Hypothesis
Isolate Bounding Box of Hand Hypothesis
Isolate Hypothesis and Skin Regions of Bounding Box
Isolate Relevant Skin Contours
Remove Empty Boundary Regions
Isolated Hand

Fig. 19 Overview of the proposed hand segmentation sub-component

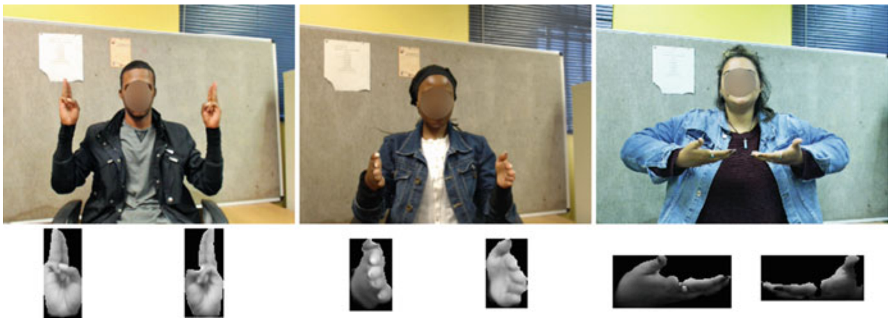


Fig. 20 Three images (top row) with the corresponding segmented hands of each image displayed beneath it

in the skin image, a selection procedure is included to strategically identify and include possibly relevant connected skin objects. This is described and illustrated in greater detail in [1]. Finally, any empty regions above, below, and to the sides of the resulting image are removed. This results in an image containing only the isolated hand. Figure 20 provides three examples of the proposed hand segmentation applied to images of three users.

5 Feature Representation

Feature representation involves normalising the final image resulting from the segmentation procedure described in the previous step by resizing it to a pre-defined width and height, followed by the application of a chosen feature descriptor to obtain a classification feature vector. The feature descriptor selected in this research is the histogram of oriented gradients (HOG) descriptor. This feature descriptor was originally proposed by Dalal and Triggs [16] and applied to pedestrian detection, with excellent results. In this, and many other applications [17–24], the feature descriptor is used to achieve very robust and high accuracy recognition.

Conceptually, the HOG feature descriptor is based on the idea that objects can be represented in terms of the distribution of the gradient contrasts in an image, such as an intensity image, containing the object. The HOG descriptor has a number of parameters that can be optimised, including: the width and height (w, h) of the input image; the number of orientations used c_o; the number of pixels c_w and c_h in each cell in the x and y dimension; and the number of cells b_w and b_h in each block in the x and y dimension. The result of the operation is a feature vector $\vec{F}$ which is passed to the recognition sub-component explained in the next section.

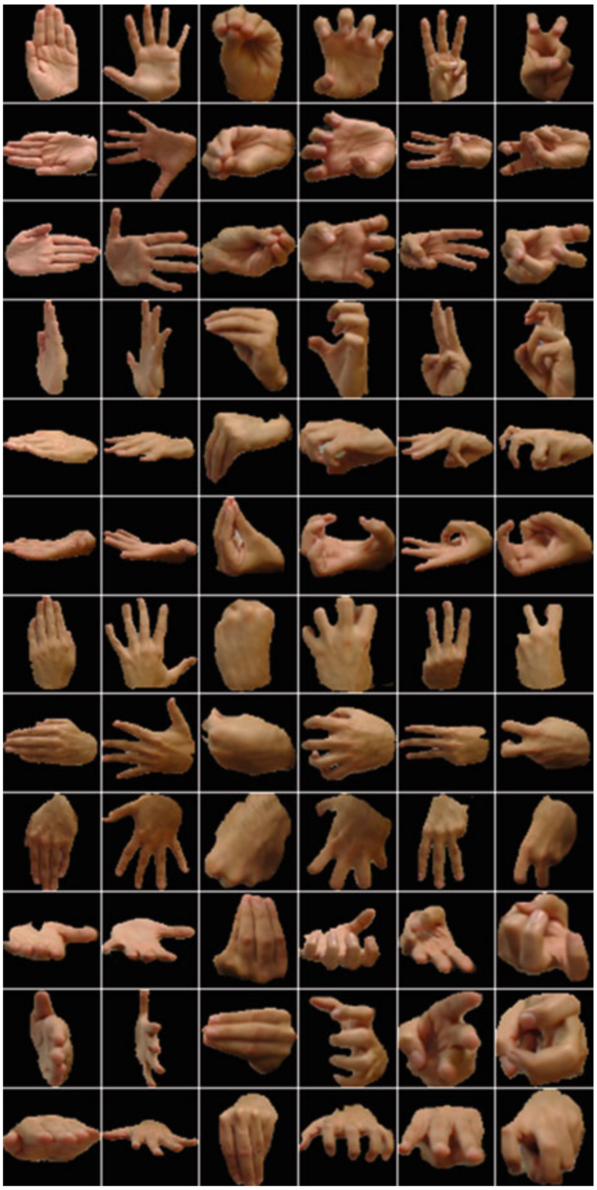


Fig. 21 The 72 combinations of the 6 hand shapes in 12 hand orientations adopted in, and recognised by, the proposed framework: columns (left to right) are hand shapes 1–6; rows (top-to-bottom) are hand orientations 1–12

6 Hand Orientation and Shape Recognition

With the feature vector $\vec{F}$ obtained, the hand orientation and hand shape recognition sub-component is invoked. A total of 6 hand shapes in 12 hand orientations were adopted from the Fulton School for the Deaf SASL dictionary [13] for inclusion in the proposed framework (see Fig. 21). A two-stage procedure is invoked to recognise the hand orientation followed by the hand shape. This is done to reduce the complexity of the classification task, which is proportional to the number of classes to be recognised.

$\vec{F}$ is first passed to a classifier that is trained to recognise the hand orientation of the hand image. In addition, a total of 12 classifiers are trained to recognise the hand shape of a hand image, each specific to a given hand orientation. The classifiers used were support vector machines (SVM). The SVM supervised machine learning algorithm was originally introduced by Vapnik [25] as a binary classification technique. However, adaptations and configurations were later proposed to cater for multi-class classification problems. It has been argued that the SVM is among the best machine learning techniques [26–28]. This is attributed to its ability to always obtain a converged model that provides superior accuracy [26] in a variety of contexts. SVMs are also a regular choice in systems that use the HOG feature descriptor such as several recent systems [22–24]. The current implementation makes use of the directed acyclic graph SVM multi-class classification technique. This technique combines the training efficiency of the one-versus-one technique with a classification efficiency that is better than the one-versus-all technique.

To arrive at trained models for these 13 SVMs, a labelled training data set of the hand orientations and hand shapes performed by several users was used as input to an SVM training and optimisation procedure. Because of the unavailability of pre-existing data sets with a suitable variety of hand shapes and orientations, a new data set was constructed using 11 test subjects representing both genders and a range of skin tones, body dimensions, and ethnicities. Using the data set, the SVMs and the HOG feature descriptor were jointly optimised. A detailed description of the data set and its construction, as well as the optimisation procedure, may be found in [1].

7 SignWriting Lookup and Transcription

At this stage in the processing workflow, the parameters that have been obtained are encoded in SWML notation [29]. This is a simple lookup and encoding process. In this research, version 1.1 of the SWML was used, but the use of newer versions can be investigated in future. A complete description of SWML is available in [29]. All references to SWML will henceforth refer to SWML 1.1.

Although it has been stated clearly that this research limits itself to producing the SWML transcription of sign language manual gestures—which has thus been achieved—it is worth mentioning that an SWML transcription can be visualised in SignWriting using the SignText Editor [30] or rendered as an actual sign language animation using a SignWriting rendering system such as the ones proposed in [31–33]. These additional functions can be incorporated directly into the proposed framework in future.

8 Summary

This chapter described the two components—the hand retrieval and the manual parameter representation and recognition components—of the proposed sign language manual parameter recognition and transcription framework. The task of the hand retrieval component is to find and track the hands in a video sequence. A complete description of the procedure and methods used to achieve tracking from video input was provided. The task of the MPRR component is to represent the motions of the hands, to recognise their orientations and shapes and represent them as SWML notation transcriptions.

The chapter described the proposed method of representing hand motions in terms of key hand locations in time. To reduce redundancy and to achieve a compact representation of the hand motions, it was devised to only retain the hand locations in key frames—frames in which the hands stopped or changed direction. It was shown that doing so provided a compact, yet fully descriptive, representation of the hand motions in time. It was also resolved to only recognise the orientation and shape of the hands in key frames. To this end, a detailed description of a model to detect holds and movements of the hands was provided.

At each key frame, hand segmentation is carried out to obtain isolated images of the hands for feature representation and recognition. A detailed description of the proposed hand segmentation procedure was provided. This was followed by an in-depth justification of, and discussion on, the HOG feature descriptor which is subsequently applied to the segmented hand images for recognition purposes.

Finally, SVMs are used to recognise the orientation and shape of the hands. This is a two-stage procedure in which the orientation of each hand is first recognised, followed by the use of one of 12 hand shape classifiers, each trained to recognise hand shapes in one of the 12 hand orientations considered. A description of the MOSH data set used to train and test these classifiers was provided. This was followed by an in-depth description of the classification mechanism of SVMs, and the procedure used to optimise them, along with a chosen feature descriptor.

At the end of this component, all four manual parameters of each hand in a *hold* state are known: the hand location, the hand motion, the hand orientation, and the hand shape. These are mapped onto, and represented in, SWML notation to produce transcriptions of the hand gestures in the input video sequence.

Acknowledgment This research is supported in part by the National Research Foundation of South Africa (UNIQUE GRANT NO: 105670).

References

1. M. Ghaziasgar, Automatic sign language manual parameter recognition. Ph.D. thesis, University of the Western Cape, Computer Science, 2017

2. A.A. Argyros, M.I. Lourakis, Real-time tracking of multiple skin-colored objects with a possibly moving camera, in *Computer Vision-ECCV 2004* (Springer, Berlin, 2004), pp. 368–379
3. A.K. Jain, *Fundamentals of Digital Image Processing* (Prentice-Hall, Upper Saddle River, 1989)
4. J.P. Lewis, Fast template matching, in *Vision Interface*, vol. 95 (1995), pp. 15–19
5. G. Bradski, A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library* (O'Reilly Media, Sebastopol, 2008)
6. P. Viola, M.J. Jones, Robust real-time face detection. Int. J. Comput. Vis. **57**(2), 137–154 (2004)
7. M.J. Taylor, T. Morris, Adaptive skin segmentation via feature-based face detection, in *Proceedings SPIE Photonics Europe* (International Society for Optics and Photonics, Bellingham, 2014), pp. 91390P–1–12
8. J.R. Smith, S.-F. Chang, Tools and techniques for color image retrieval, in *Electronic Imaging: Science & Technology* (International Society for Optics and Photonics, Bellingham, 1996), pp. 426–437
9. N. Otsu, A threshold selection method from gray-level histograms. Automatica **11**(285–296), 23–27 (1975)
10. C. Stauffer, W. Grimson, Adaptive background mixture models for real-time tracking, in *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2 (IEEE, Piscataway, 1999), pp. 246–252
11. Z. Zivkovic, Improved adaptive Gaussian mixture model for background subtraction, in *Proceedings 17th International Conference on Pattern Recognition*, vol. 2 (IEEE, Piscataway, 2004), pp. 28–31
12. I. Achmed, I.M. Venter, P. Eisert, Improved hand-tracking framework with a recovery mechanism, in *Proceedings South African Telecommunication Networks and Applications Conference, Spier* (2013), pp. 344–349
13. S. Howard, T. Chowles, L. Reynolds, *Finger Talk: South African Sign Language (SASL) Dictionary* (Fulton School for the Deaf, Gillitts, 2011)
14. M. Raptis, L. Sigal, Poselet key-framing: a model for human activity recognition, in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 2650–2657
15. J. Tilmanne, S. Hidot, T. Ravet, MocKey: motion capture as a tool for keyframing animation. QPSR Numediart Res. Program **2**(4), 119–124 (2009)
16. N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1 (IEEE, Piscataway, 2005), pp. 886–893
17. O. Lanihun, B. Tiddeman, E. Tuci, P. Shaw, Improving active vision system categorization capability through histogram of oriented gradients, in *Conference Towards Autonomous Robotic Systems* (Springer, Cham, 2015), pp. 143–148
18. S. Tian, U. Bhattacharya, S. Lu, B. Su, Q. Wang, X. Wei, Y. Lu, C.L. Tan, Multilingual scene character recognition with co-occurrence of histogram of oriented gradients. Pattern Recogn. **51**, 125–134 (2016)
19. I.L. Bastos, M.F. Angelo, A.C. Loula, Recognition of static gestures applied to Brazilian sign language (Libras), in *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images* (IEEE, Piscataway, 2015), pp. 305–312
20. O. Déniz, G. Bueno, J. Salido, F. De la Torre, Face recognition using histograms of oriented gradients. Pattern Recogn. Lett. **32**(12), 1598–1603 (2011)
21. P.E. Rybski, D. Huber, D.D. Morris, R. Hoffman, Visual classification of coarse vehicle orientation using histogram of oriented gradients features, in *Intelligent Vehicles Symposium (IV), 2010 IEEE* (IEEE, Piscataway, 2010), pp. 921–928
22. Y. Yang, L. Lin, Automatic pedestrians segmentation based on machine learning in surveillance video, in *2019 IEEE International Conference on Computational Electromagnetics (ICCEM)* (IEEE, Piscataway, 2019), pp. 1–3

23. C. Wang, Z. Li, N. Dey, Z. Li, A.S. Ashour, S.J. Fong, R.S. Sherratt, L. Wu, F. Shi, Histogram of oriented gradient based plantar pressure image feature extraction and classification employing fuzzy support vector machine. J. Med. Imaging Health Informatics **8**(4), 842–854 (2018)
24. R. Kapoor, R. Gupta, S. Jha, R. Kumar et al., Detection of power quality event using histogram of oriented gradients and support vector machine. Measurement **120**, 52–75 (2018)
25. C. Cortes, V. Vapnik, Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
26. S.S. Rautaray, A. Agrawal, Vision based hand gesture recognition for human computer interaction: a survey. Artif. Intell. Rev. **43**(1), 1–54 (2015)
27. S. Sharma, C.R. Krishna, S.K. Sahay, Detection of advanced malware by machine learning techniques, in *Soft Computing: Theories and Applications* (Springer, Singapore, 2019), pp. 333–342
28. H. Faris, M.A. Hassonah, A.-Z. Ala'M, S. Mirjalili, I. Aljarah, A multi-verse optimizer approach for feature selection and optimizing SVM parameters based on a robust system architecture. Neural Comput. Appl. **30**(8), 2355–2369 (2018)
29. A.C. da Rocha Costa, G.P. Dimuro, Signwriting and SWML: paving the way to sign language processing, in *TALN 2003* (2003), pp. 193–202
30. S. Slevinski, SignText editor. Available at http://www.signbank.org/signpuddle/signtext/signtext.html
31. K. Abrahams, M. Ghaziasgar, J. Connan, R. Dodds, Rendering South African sign language sentences from SignWriting notation, in *Proceedings South African Telecommunication Networks and Applications Conference, Port Elizabeth* (2014), pp. 87–92
32. D. Bragg, R. Kushalnagar, R. Ladner, Designing an animated character system for American sign language, in *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility* (ACM, New York, 2018), pp. 282–294
33. A. Balayn, H. Brock, K. Nakadai, Data-driven development of virtual sign language communication agents, in *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)* (IEEE, Piscataway, 2018), pp. 370–377

Computer Vision Algorithms for Image Segmentation, Motion Detection, and Classification

Mehrdad Ghaziasgar, Antoine Bagula, and Christopher Thron

1 Introduction

This chapter describes a number of widely used techniques in computer vision, which are also used in the automatic sign language recognition system described in [1]. These techniques are categorised as follows:

- Image segmentation
  - Edge detection via adaptive Gaussian thresholding and image inversion;
  - Cross correlation template matching for shape recognition.
  - Viola–Jones face detection;
- Motion detection and tracking:
  - Gaussian mixture modeling.
- Classification:
  - Histogram of oriented gradients (HOG) descriptor
  - Support vector machine (SVM) classification;
  - Directed acyclic graph multi-class SVM.

M. Ghaziasgar (✉) · A. Bagula
Department of Computer Science, University of the Western Cape, Cape Town, South Africa
e-mail: mghaziasgar@uwc.ac.za

C. Thron
Department of Science and Mathematics, Texas A&M University-Central Texas, Killeen, TX, USA

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_5

2 Image Segmentation

"Image segmentation" refers to the identification of image regions of particular interest. Such regions may be identified by means of particular characteristics such as edges, shapes, contrasts, and so on [2]. In this section we present three techniques that may be used in such identifications: adaptive Gaussian thresholding and image inversion are used to locate edges that may bound regions of interest; cross correlation template matching is used to pick out certain shapes in the image; and Viola–Jones face detection is a specialised module that looks specifically for image portions that are likely to be faces. These three techniques are described in the following subsections.

2.1 Adaptive Gaussian Thresholding and Image Inversion

A thresholding function S converts a grayscale image L into a binary image by applying a threshold ϵ to the image. The value of the pixel $L(x, y)$ at (x, y) in the grayscale image is compared to ϵ and set to white, i.e., 255, if it exceeds or equals ϵ, or black, i.e., 0, otherwise. This is formally given by

$$S(x, y) = \begin{cases} \text{White} & \text{if} L(x, y) \geq \epsilon \\ \text{Black} & \text{if} L(x, y) < \epsilon \end{cases} \tag{1}$$

Note that, due to the (x, y) indexing, S can be viewed as the resulting thresholded image. The value of ϵ can be fixed or adaptive [3]. If it is adaptive, it takes the form of a function $\epsilon(x, y)$ which is specific to each pixel. With adaptive Gaussian thresholding [2], $\epsilon(x, y)$ is computed as the difference between the weighted sum of a $b \times b$ neighbourhood around each pixel (x, y) and a constant K. The weights in the neighbourhood are assigned by a Gaussian window G of size $b \times b$ pixels given by

$$G(i, j) = \frac{1}{2\pi\sigma^2} e^{\frac{i^2+j^2}{2\sigma^2}}; \quad -(b-2) \leq i \leq (b-2); -(b-2) \leq j \leq (b-2) \tag{2}$$

where i and j are indices in the window G and σ is the standard deviation of the Gaussian distribution typically taken as 1. Therefore, assuming that the $b \times b$ neighbourhood around the pixel at (x, y) in L is $B(x, y)$, $\epsilon(x, y)$ is given by

$$\epsilon(x, y) = B(x, y) \star G - K \tag{3}$$

Applying this function to a grayscale image such as the one in Fig. 1a results in a binary image in which regions of the input image with little or no change in intensity are highlighted and regions with strong changes in intensity are darkened,

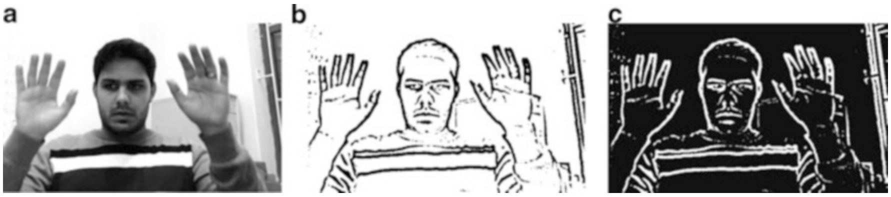


Fig. 1 Adaptive Gaussian thresholding and image inversion: (**a**) the grayscale image of the input; (**b**) the corresponding thresholded image with strong intensity changes darkened using adaptive Gaussian thresholding; (**c**) the inverted image in which strong intensity changes, i.e., edges, are highlighted

Fig. 2 Contour hand templates: (**a**) right hand; (**b**) left hand

as illustrated in Fig. 1b. A simple inversion of this image results in an image in which strong intensity changes, i.e., edges, are highlighted, as illustrated in Fig. 1c.

The size of the neighbourhood used must be small enough to maintain the pixel-level adaptive quality of the technique, but large enough to incorporate information from a local neighbourhood. Set too small, the contours produced are very dense and represent very small-scale edges. Set too large, the approach loses its adaptive quality and becomes a variant of global thresholding. Practically speaking, any value of $9 \leq b \leq 21$ provides relatively good contour detection results. In this implementation, a neighbourhood size of $b = 11$ was used to obtain contours.

2.2 *Cross Correlation Template Matching*

Cross correlation template matching is used to find the most likely position of a shape with known outline within an image, which is specified by a contour template. Examples of contour templates for hand shapes are shown in Fig. 2. The matching algorithm for a given shape involves computing statistical correlations between the shape's fixed template T and the search image S. A search window of the same width w and height h as T is passed over S. At each position (x, y), the correlation C is computed as follows. Given that (i, j) and (a, b) are coordinate index counters in T such that $i, a \in \{1, \ldots, w\}$ and $j, b \in \{1, \ldots, h\}$, $C(x, y)$ is given by

$$C(x, y) = \sum_{i,j} [T'(i, j) \cdot S'(x + i, y + j)]^2 \tag{4}$$

where

$$T'(i, j) = T(i, j) - \frac{1}{(w \cdot h) \sum_{a,b} T(a, b)} \tag{5}$$

$$S'(x+i, y+j) = S(x+i, y+j) - \frac{1}{(w \cdot h) \sum_{a,b} I(x+a, y+b)} \tag{6}$$

The position (x, y) corresponding to the maximum value of $C(x, y)$ is the most likely position of the shape represented by T. It may also be a false positive. Therefore, an empirically determined correlation threshold t_c determines whether $C_{\max}(x, y)$ represents a true or false positive detection. Note that this procedure is carried out independently for each template used.

2.3 Viola–Jones Face Detection

The Viola–Jones face detection framework [4] is used to detect and locate the user's face. It is a very efficient and accurate face detection approach. For this reason, it is perhaps the most prominent face detection approach in the literature [5].

Generally speaking, the Viola–Jones face detection framework uses fundamental Haar-like wavelet features (or Haar-like features in short) in a grayscale image to locate one or more faces. The Haar-like features are of specific types but can take on a variety of sizes. Computing these features in an image can be a costly procedure. Viola and Jones proposed an image representation—the Integral image—that can be used to efficiently compute these features at any scale in an image. Finally, a modified AdaBoost classifier is used to arrange a series of weak classifiers, each trained on a single Haar-like wavelet feature, into a face detection rejection cascade. The rejection cascade determines whether or not a given input is a face. The location of the face in the input image is thus determined. These steps are explained in greater detail in the subsections below.

2.3.1 Haar-Like Wavelet Features and Their Computation

Haar-like features are characterised by various unique arrangements of alternating light and dark rectangles, either vertically or horizontally adjacent. Five example Haar-like features are provided in Fig. 3. Note that other possibilities also exist such as those resulting from inter-changing the light and dark regions of each feature in the figure.

Each of the features is computed at every feasible location and scale in a grayscale image: starting at the largest or smallest scale, the feature is scanned over

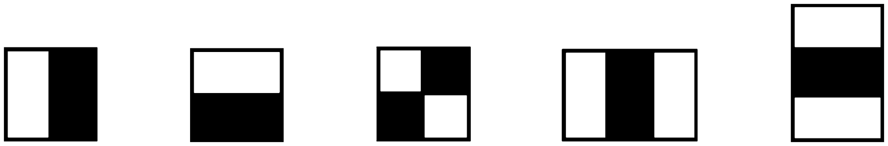


Fig. 3 Example Haar-like wavelet features

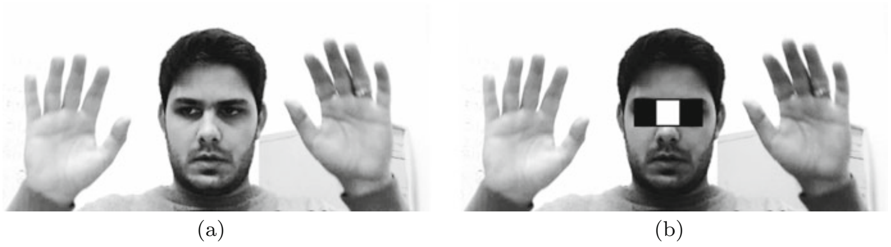


Fig. 4 Example of a Haar-like feature on an image: (**a**) grayscale image showing that the eyes are generally darker than the bridge of the nose; (**b**) the Haar-like feature placed over the eyes

the image and computed at each location. Once all possibilities at this scale are exhausted, this process is repeated for the next scale.

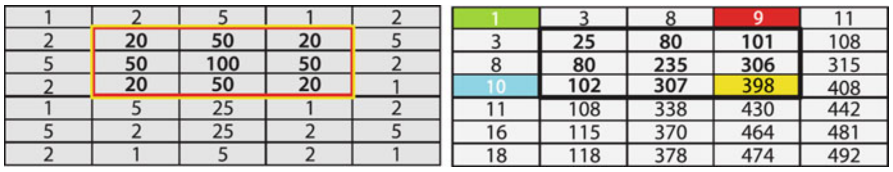
Computing the value of a specific feature at a given location and scale amounts to computing the sum of the image pixel values corresponding to the dark region(s) of the feature and subtracting this sum from the sum of the image pixel values corresponding to the light region(s) of the feature. The feature is then said to be present at that scale and location if its value exceeds a pre-determined threshold.

Figure 4 provides an example of a feature placed over the eyes of the user. The feature used is a horizontal feature with three blocks; two dark blocks on the sides and one light block in the middle. Computing the value of this feature at this location and scale amounts to subtracting the sum of the pixels in the eye regions from the sum of the pixels on the bridge of the nose. It is clear from Fig. 4a that the eyes are darker than the bridge of the nose. Therefore, it is very likely that the value of this feature at this location and scale will exceed the detection threshold, and thereby be determined to be present.

2.3.2 Integral Image Representation for Haar-Like Wavelet Computation

Computing the sum of pixels of various features at a variety of scales and locations can be very computationally costly. Viola and Jones proposed an image representation—the Integral image—which provides a very efficient means of computing pixel sums at every scale and location.

Letting the original grayscale image be L, the value at each location (x, y) in the Integral image L' is the sum of the pixel itself i.e., $L(x, y)$, and all pixels above and



1	2	5	1	2
2	20	50	20	5
5	50	100	50	2
2	20	50	20	1
1	5	25	1	2
5	2	25	2	5
2	1	5	2	1

(a)

1	3	8	9	11
3	25	80	101	108
8	80	235	306	315
10	102	307	398	408
11	108	338	430	442
16	115	370	464	481
18	118	378	474	492

(b)

Fig. 5 Example of Integral image computation, adapted from [3]: (**a**) the values in an original grayscale image; (**b**) the corresponding Integral image

to the left of that pixel location. This can be efficiently represented and computed using the following recurrence relation [4]:

$$L'(x, y) = L(x, y) + L'(x - 1, y) + L'(x, y - 1) - L'(x - 1, y - 1) \quad (7)$$

Computing the Integral image L' requires one pass through the image L. Using L', it is possible to compute the sum inside any rectangle in L with only a small number of lookups in L'. For example, the sum of pixels in the highlighted rectangle in Fig. 5a can be computed from the values in the corresponding Integral image in Fig. 5b as follows. Referring to the values in Fig. 5b, the values in the red and blue cells—9 and 10—are subtracted from the value in the yellow cell—398—and the value in the green cell—1—is added to the result. The sum of pixels in the highlighted box in Fig. 5a—380—is thus efficiently obtained.

Using a similar approach, the value of any feature at any scale and location can be determined using a small number of lookups.

2.3.3 Selection of Features Using AdaBoost and Arrangement into a Rejection Cascade

Given the variety of features that can be used, an important element of the Viola–Jones face detector is the use of a modified AdaBoost learning algorithm to select features which best distinguish between negative and positive face examples in a training set.

A set of classifiers are trained on each of the features selected. Note that these classifiers are weak classifiers since each classifier, on its own, is only able to determine the presence of a specific feature, and not the presence of a face. The weak classifiers are placed into a rejection cascade illustrated in Fig. 6 in descending order of their importance towards face detection, as determined by the AdaBoost algorithm.

Given any sub-window in an image, each classifier in the cascade is invoked in order, starting with the first and most relevant classifier. If any classifier rejects the sub-window, i.e., that feature is not present, processing on the sub-window stops immediately and moves on to a different sub-window. It is determined that

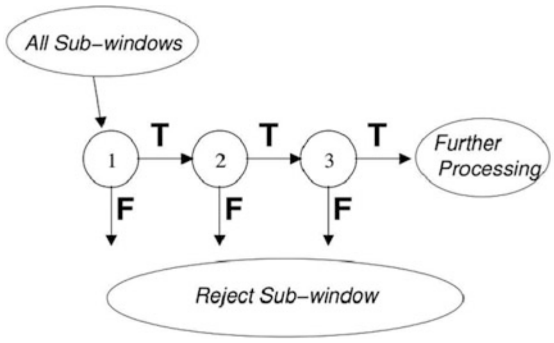


Fig. 6 An illustration of a rejection cascade of weak classifiers [4]

a face is not present in this sub-window. This approach significantly reduces the computational overhead of the algorithm and ensures that minimal time is wasted on non-face background regions. It is only determined that a face is present if and when every classifier in the cascade accepts the sub-window.

3 Motion Detection Using Gaussian Mixture Modeling

Motion detection is a key aspect in several computer vision applications that involve sensing and/or surveillance. Motion detection involves examining the images in a sequence of consecutive images of the same scene, and partitioning the pixels in each image into *background* and *foreground*, where the foreground pixels belong to moving objects. Identification of foreground pixels in an image involves finding pixels that are “significantly different” from the same pixel location in previous images. The notion of “significance” depends on the particular application: users will want to screen out changes due to lighting variations, windblown motion, camera jiggling, and so on.

A wide variety of motion detection algorithms exist: for a survey, see [6, 7]. In this section, we will describe one particular motion detection algorithm, as developed and implemented in [8]. This algorithm relies on Gaussian mixture modeling, which is a key component of many motion detection algorithms.

We consider one particular pixel (i, j) (all pixels are treated the same) and denote the intensity vector in colour space of the pixel at time index t as $\vec{I}_t$. The pixel’s probability distribution is modeled by a mixture of k Gaussians, so that the probability that pixel (i, j) takes the value $\vec{v}$ at time index t is given by

$$P(\vec{I}_t = \vec{v}) = \sum_{n=1}^{k} W_{n,t} \cdot \mathcal{N}(\vec{v}; \vec{\mu}_{n,t}, \Sigma_{n,t}) \tag{8}$$

where $W_{n,t}$ is the estimated weight parameter of the n-th Gaussian component, and $\mathcal{N}$ is given by

$$\mathcal{N}(v; \mu_{n,t}, \sigma_{n,t}) = \frac{1}{(2\pi)^{\frac{n}{2}} \mid \Sigma_{n,t} \mid^{\frac{1}{2}}} e^{-\frac{1}{2}(\vec{v}-\vec{\mu}_{n,t})^T \Sigma_{n,t}^{-1}(\vec{v}-\vec{\mu}_{n,t})} \tag{9}$$

where $\vec{\mu}_{n,t}$ and $\Sigma_{n,t}^2 = \sigma_{n,t}^2 \mathbf{I}$ are the mean and covariance, respectively, of the n-th Gaussian component. The expression $\frac{W_{n,t}}{\sigma_{n,t}}$ is known as the *fitness value*, and is used to order the k distributions. The first B components are taken as a probabilistic model of the background, where the parameter B is determined by

$$B = \text{argmin}_b \left(\sum_{n=1}^{b} W_{n,t} > T \right) \tag{10}$$

and T is a user-defined threshold.

The pixel measurement I_t at time index t is deemed to be consistent with the n-th cluster if it lies within 2.5 standard deviations of the cluster's mean, i.e., $|\vec{I}_t - \vec{\mu}_{n,t}| \leq 2.5\sigma_{n,t}$. If the pixel is not consistent with any of the B background components, then the pixel is designated as foreground, and is grouped with other foreground pixels using 2D connected component analysis to determine which moving object it belongs to.

Based on the measured value I_t, the n Gaussian components are updated as follows. If the value of I_t is consistent with the n-th Gaussian component, then the n-th mean and variance are updated as shown below:

$$W_{n,t} = W_{n,t-1} \tag{11a}$$

$$\mu_{n,t} = (1-\rho)\mu_{n,t-1} + \rho I(i,j,t) \tag{11b}$$

$$\sigma_{n,t}^2 = (1-\rho)\sigma_{n,t-1}^2 + \rho(I(i,j,t) - \mu_{n,t})^2 \tag{11c}$$

$$\rho = \alpha\mathcal{N}(I(i,j,t) \mid \mu_k, \Sigma_k) \tag{11d}$$

where $\frac{1}{\alpha}$ is a user-chosen time constant that reflects the time scale over which background features are expected to be unchanged. This update amounts to changing the mean and covariance of the component based on information supplied by the new measurement. If on the other hand the n-th Gaussian component is not consistent with I_t, then the expression below is used:

$$W_{n,t} = (1-\alpha)W_{n,t-1} \tag{12a}$$

$$\mu_{n,t} = \mu_{n,t-1} \tag{12b}$$

$$\sigma_{n,t}^2 = \sigma_{n,t-1}^2 \tag{12c}$$

In this case the component's mean and covariance are not changed, because the measurement is supposed to be due to another object coming in front which belongs to a different component.

If none of the Gaussian components are consistent with the measured vector I_t, then a new Gaussian component is defined with a low weight parameter, high variance, and mean equal to I_t. This new component then replaces the component with lowest fitness.

4 Feature Representation Using the Histogram of Oriented Gradients Feature Descriptor

The histogram of oriented gradients (HOG) descriptor was invented in 1986 by McConnell [9] and popularised by Dalal and Triggs [10]. HOG has been used in conjunction with support vector machines in a variety of research studies very recently [11–13]. In [1], HOG is used in the classification of hand gestures. In this section we give a step-by-step of the construction of the HOG feature vector.

HOG begins with a preprocessed grayscale image of the object of interest. Preprocessing usually involves segmentation to isolate the object, as well as resizing to a pre-defined width and height ($w \times h$), resulting in an input image $\hat{I}$.

The x and y components $g_x(x, y)$ and $g_y(x, y)$ of the image intensity gradient $g(x, y)$ at each pixel (x, y) in $\hat{I}$ are computed by means of a 1D-centred masking operation $[-1, 0, 1]$ in the respective x and y directions, given by:

$$g_x(x, y) = \hat{I}(x + 1, y) - \hat{I}(x - 1, y) \tag{13a}$$

$$g_y(x, y) = \hat{I}(x, y + 1) - \hat{I}(x, y - 1) \tag{13b}$$

where $x \in \{1, \dots, w\}$ and $y \in \{1, \dots, h\}$. For the bordering pixels of $\hat{I}$ for which $x = 1, x = w, y = 1$, or $y = h$, the undefined values of $\hat{I}$ in Eq. 13 are assumed to be zero. The gradient components g_x and g_y are used to compute the gradient magnitude $\boldsymbol{m}(x, y)$ and gradient orientation $\boldsymbol{\phi}(x, y)$ at each pixel (x, y) as follows:

$$\boldsymbol{m}(x, y) = \sqrt{g_x(x, y)^2 + g_y(x, y)^2} \tag{14a}$$

$$\boldsymbol{\phi}(x, y) = \tan^{-1}\left(\frac{g_y(x, y)}{g_x(x, y)}\right) \tag{14b}$$

The unsigned equivalent of $\boldsymbol{\phi}$ limits contrast gradients to the range $[0, \pi]$ as follows:

$$\boldsymbol{\phi}_U(x, y) = \begin{cases} \boldsymbol{\phi}(x, y) & \text{if } \boldsymbol{\phi}(x, y) \geq 0 \\ \boldsymbol{\phi}(x, y) + \pi & \text{if Otherwise} \end{cases} \tag{15}$$

$\hat{I}$ is then partitioned into cells by means of a grid, with the width and height of each cell being $(c_w \times c_h)$ pixels. For reference, cells in the grid are indexed as (i, j), where $i \in \left\{1, \ldots, \frac{w}{c_w}\right\}$ and $j \in \left\{1, \ldots, \frac{h}{c_h}\right\}$. In each cell at (i, j), the gradient orientations are quantised into c_o orientation bins and a histogram $H(i, j)$ thereof is computed. The orientation vote $\boldsymbol{\phi}_U(x, y)$ of each pixel in the cell is the magnitude $\boldsymbol{m}(x, y)$ of that pixel. Formally, the value of the orientation bin o in the histogram $H(i, j)$ corresponding to the cell C at index (i, j) is given by $H(i, j, o)$ as follows:

$$H(i, j, o) = \sum_{x,y \in C} h\Big(\boldsymbol{\phi}_U(x, y) = o, \quad \boldsymbol{m}(x, y)\Big) \tag{16}$$

for all $o \in \{1, \ldots, c_o\}$ and

$$h(A, v) = \begin{cases} v & \text{if } A \text{ is True} \\ 0 & \text{if } A \text{ is False} \end{cases} \tag{17}$$

In the simple case, the final feature vector resulting from the HOG feature descriptor can be taken as the concatenation of all histograms $H(i, j)$ in $\hat{I}$. However, Dalal and Triggs proposed to further group cells into overlapping blocks and produce normalised versions of the histograms in the cells of each block. This not only represents larger-scale features in the image better, but also significantly increases robustness to variations in illumination [10].

Accordingly, cells are further grouped into overlapping blocks, with the width and height of each block being $(b_w \times b_h)$ cells. The overlap of blocks depends on b_w and b_h, where the overlap in the x and y directions, respectively, are $b_w - 1$ and $b_h - 1$ cells. For ease of reference, new indices are introduced to index the cells in a specific block, where the histograms of cells in a specific block b are indexed as $H_b(h, k)$ for $h \in \{1, \ldots, b_w\}$, $k \in \{1, \ldots, b_h\}$, $b \in \{1, \ldots, B\}$ and B is the number of blocks.

Using the histograms $H_b(h, k)$ within each block produced previously, a new set of normalised histograms $\widehat{H_b}(h, k)$ are produced and assigned to this specific block b. The normalisation is a simple one involving dividing each histogram value by the block sum. To avoid division-by-zero errors, a small value ϵ is added to the block sum, and the square root of their squares is computed. As before, assuming the value of the orientation bin o in the histogram $H_b(h, k)$ corresponding to the cell at index (h, k) is referred to as $H(h, k, o)$, the block sum S_b of a specific block b is given by:

$$S_b = \sum_{h,k,o} H_b(h, k, o) \tag{18}$$

for all (h, k) in block b and $o \in \{1, \ldots, c_o\}$. The normalised histogram $\widehat{H_b}(h, k)$ is then given by:

$$\widehat{H}_b(h,k) = \frac{H_b(h,k)}{\sqrt{S_b^2 + \epsilon^2}} \tag{19}$$

Although the overlap between blocks appears to introduce redundancy, it is important to note that the $\widehat{H}_b(h,k)$ version of each cell histogram is adapted to the specific block that it appears in and is most probably different from the original $H_b(h,k)$. The final HOG feature vector $\vec{F}$ is a concatenation of all histograms $\widehat{H}_b(h,k)$ for all blocks $b \in \{1,\ldots,B\}$. The size of this feature vector is the product of the number of blocks, the number of cells per block in the x direction b_w, the number of cells per block in the y direction b_h, and the number of orientations used c_o: $\left[\left(\frac{w}{c_w} - b_w + 1\right) \cdot \left(\frac{h}{c_h} - b_h + 1\right)\right] \cdot b_w \cdot b_h \cdot c_o$.

Once the feature vector $\vec{F}$ has been computed, it may be subjected to a classification algorithm. With HOG, the most commonly used classifier is the support vector machines classifier, which is discussed in the next section.

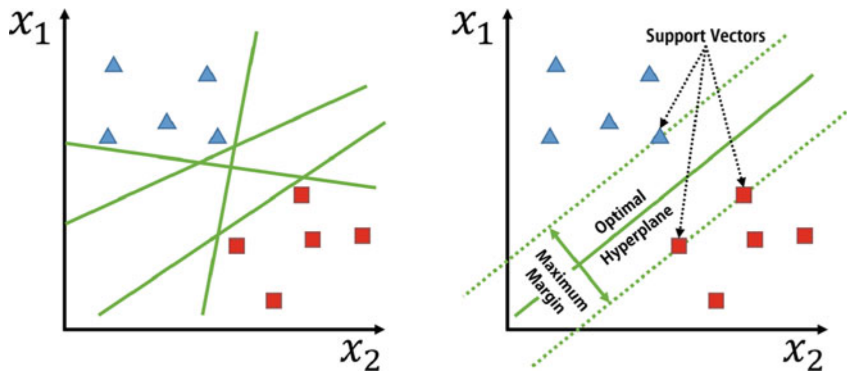
5 Support Vector Machine Classification

Together with artificial neural networks, support vector machines (SVM) ranks as one of the most powerful tools for classification in the modern practice of machine learning. In the following subsections, we will explain the SVM and the different types of SVM in more detail.

5.1 Support Vector Machine Classification Principle

Consider the set of points shown in Fig. 7a. The figure depicts a two-class classification problem. The red square points and blue triangular points are both expressed in terms of features x_1 and x_2, and belong to two separate intended classes. In general, the aim of classification is to obtain a hyperplane that separates the two classes of points. Then, given a new data point expressed in terms of x_1 and x_2, the point is assigned to one of the two classes depending on which side of the hyperplane it falls.

Figure 7a makes it clear that many different separating hyperplanes can be used to separate the two classes. The principal idea of SVM classification is to determine the hyperplane that provides the widest separation or "maximum margin" between the two classes. This hyperplane is referred to as the "optimal hyperplane" and is depicted in Fig. 7b. Ensuring that the maximum margin is maintained helps provide the optimal separation between the classes, thereby reducing misclassification of unseen data points in run-time.



(a) (b)

Fig. 7 A two-class classification problem: (**a**) various (green) hyperplanes that can be used to separate the two classes; (**b**) the optimal hyperplane—the hyperplane that separates the two classes with the largest margin between the classes

A decision rule is formulated below to conform to the decision boundary which meets the requirement of a maximum margin. Let the set of N points be $X = \{x_i | i = 1, \ldots, N\}$, noting that each x_i is a point $(x_i^1, x_i^2, \ldots, x_i^M)$ with M components, where $N = 10$ and $M = 2$ in Fig. 7. The points in X can be assigned to one of two classes, a positive class C^+ and a negative class C^-, each corresponding to one of the two classes in the figure.

In the simplest terms, given an arbitrary point x that forms a vector $\vec{u}$ from the origin and lies at an arbitrary location in the Cartesian plane, the classification problem of determining the class of x amounts to determining on which side of the optimal hyperplane—or simply "hyperplane" for short—the point lies. Let $\vec{w}$ be a vector perpendicular to the hyperplane. Then for some constant C that depends on the hyperplane [14]:

$$\vec{u} \cdot \vec{w} < C \quad \text{if} \quad x \in C^- \tag{20a}$$

$$\vec{u} \cdot \vec{w} \geq C \quad \text{if} \quad x \in C^+ \tag{20b}$$

Carrying out a restructure of Eq. 20 and introducing b where $b = -C$ as a matter of convenience leads to:

$$\vec{u} \cdot \vec{w} + b < 0 \quad \text{if} \quad x \in C^- \tag{21a}$$

and

$$\vec{u} \cdot \vec{w} + b \geq 0 \quad \text{if} \quad x \in C^+ \tag{21b}$$

Assuming that $\vec{x}^-$ and $\vec{x}^+$ are, respectively, arbitrary negative and positive samples in X, a rescale of $\vec{w}$ leads to:

$$\vec{x}^- \cdot \vec{w} + b \leq -1 \tag{22a}$$

and

$$\vec{x}^+ \cdot \vec{w} + b \geq 1 \tag{22b}$$

A variable y_i is introduced to unify Eqs. 22a and 22b and to obtain a combined generic equation for the negative and positive classes. The variable y_i is part of a set of labels $Y = \{y_i | i = 1, \ldots, N, y_i \in \{1, -1\}\}$, with each label y_i assigned to each point x_i, where $y_i = -1$ if $x_i \in C^-$ and $y_i = 1$ if $x_i \in C^+$ for all $i \in \{1, \ldots, N\}$. Multiplying y_i into either Eq. 22a or Eq. 22b leads to the same expression as below:

$$y_i(\vec{x_i} \cdot \vec{w} + b) \geq 1 \quad \forall \ x_i \in \{C^+, C^-\} \tag{23}$$

The special points x_i that lie on the boundaries of the margin on either side of the decision boundary are referred to as support vectors. The boundaries of the margin are visualised in Fig. 7b as dotted green lines and the support vectors are clearly indicated in that figure. For these points, Eq. 23 is given by

$$y_i(\vec{x_i} \cdot \vec{w} + b) = 1 \tag{24}$$

As mentioned before, the aim of SVM classification is to separate the positive and negative samples by the widest margin. Therefore, the size of the margin D is formulated by devising a vector formed by taking the difference between a support vector of the positive class $\vec{x}^+$ and a support vector of the negative class $\vec{x}^-$ and projecting this vector onto a unit vector perpendicular to the separating hyperplane. This is formulated as below:

$$\begin{aligned} D &= (\vec{x}^+ - \vec{x}^-) \cdot \frac{\vec{w}}{||w||} \\ &= \frac{\vec{w} \cdot \vec{x}^+ - \vec{w} \cdot \vec{x}^-}{||w||} \end{aligned} \tag{25}$$

Using Eq. 24 to substitute for the expressions $\vec{w} \cdot \vec{x}^+$ and $\vec{w} \cdot \vec{x}^-$ in Eq. 25 results in:

$$\begin{aligned} D &= \frac{1 - b + 1 + b}{||w||} \\ &= \frac{2}{||w||} \end{aligned} \tag{26}$$

Therefore, maximising the margin amounts to maximising Eq. 26 which, in turn, amounts to minimising the inverse of that equation subject to Eq. 24 as follows:

$$\begin{aligned} \max \frac{2}{||w||} &\implies \min \frac{||w||}{2} \\ &\implies \min ||w|| \\ &\implies \min \frac{1}{2}||w||^2 \end{aligned} \tag{27a}$$

subject to :

$$y_i(\vec{x}_i \cdot \vec{w} + b) = 1 \tag{27b}$$

The maximisation in Eq. 27a can be achieved using Lagrange multipliers. Let L be the boundary to be maximised by subtracting Eq. 27a from a summation of all the constraints found in Eq. 24 as follows:

$$L = \frac{1}{2}||\vec{w}||^2 - \sum_i^N \alpha_i [y_i(\vec{w} \cdot \vec{x}_i + b) - 1] \tag{28}$$

Differentiating L with respect to $\vec{w}$ yields the minimisation expression:

$$\begin{aligned} \frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_i^N \alpha_i y_i \vec{x}_i &= 0 \\ \vec{w} &= \sum_i^N \alpha_i y_i \vec{x}_i \end{aligned} \tag{29}$$

This expression makes it clear that $\vec{w}$ is the linear sum of all of the samples in X along with their corresponding class labels. Differentiating L with respect to b yields the minimisation expression:

$$\begin{aligned} \frac{\partial L}{\partial b} = -\sum_i^N \alpha_i y_i &= 0 \\ \sum_i^N \alpha_i y_i &= 0 \end{aligned} \tag{30}$$

Now substituting for $\vec{w}$ in Eq. 28 using Eq. 29 leads to:

$$L = \frac{1}{2}\Big(\sum_i^N \alpha_i y_i \vec{x}_i\Big) \cdot \Big(\sum_j^N \alpha_j y_j \vec{x}_j\Big) - \Big(\sum_i^N \alpha_i y_i \vec{x}_i\Big) \cdot \Big(\sum_j^N \alpha_j y_j \vec{x}_j\Big) - b\sum_i^N \alpha_i y_i + \sum_i^N \alpha_i \tag{31}$$

Substituting the expression in Eq. 30 into Eq. 31 allows for the Lagrangian to be rewritten as follows:

$$L = \sum_{i}^{N} \alpha_i - \frac{1}{2} \sum_{i}^{N} \sum_{j}^{N} \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \tag{32}$$

The formulation for the discriminant of the optimal hyperplane is therefore:

$$f(\vec{x}) = \sum_{i \in V} \alpha_i y_i \vec{x_i} \cdot \vec{x} + b \tag{33}$$

where V is the set of indices of the support vectors in X and:

$$x \in C^+ \quad \text{if} \quad f(x) \geq 0, \tag{34a}$$

$$x \in C^- \quad \text{if} \quad f(x) < 0 \tag{34b}$$

5.2 *Mapping onto Higher-Dimensional Spaces*

In many cases, the data points of the two classes C^+ and C^- are not linearly separable. In these cases, a higher-dimensional embedding "trick" called the "kernel trick" is used to map the data onto a higher-dimensional space in which the data is linearly separable [14].

In this case, the vectors $\vec{x_i}$ and $\vec{x}$ of Eq. 33 are replaced by $\phi(\vec{x_i})$ and $\phi(\vec{x})$ where ϕ is a function that maps those vectors onto a desired higher-dimensional space. Equivalently, Eq. 33 becomes

$$f(\vec{x}) = \sum_{i \in V} \alpha_i y_i \phi(\vec{x_i}) \cdot \phi(\vec{x}) + b \tag{35}$$

Given that the inner product $\phi(\vec{a}) \cdot \phi(\vec{b})$ for two vectors $\vec{a}$ and $\vec{b}$ can be expressed as a function $K(\vec{a}, \vec{b})$, Eq. 35 becomes

$$f(\vec{x}) = \sum_{i \in V} \alpha_i y_i K(\vec{x_i}, \vec{x}) + b \tag{36}$$

where K is known as a kernel function. A variety of kernel functions can be used with varied success. Four common kernel functions are as follows:

1. Linear Kernel: $K(\vec{x_i}, \vec{x}) = (\vec{x_i}) \cdot (\vec{x})$
2. Radial Basis Function (RBF) Kernel: $K(\vec{x_i}, \vec{x}) = \exp(-\gamma(||\vec{x_i} - \vec{x}||_2^2))$
3. Polynomial Kernel: $K(\vec{x_i}, \vec{x}) = (\gamma(\vec{x}) \cdot (\vec{x}) + z)^d$
4. Sigmoid Kernel: $K(\vec{x_i}, \vec{x}) = \tanh(\gamma(\vec{x_i}) \cdot \vec{x} + b)$

where γ, z, and d are the kernel parameters. The choice of kernel used is important as it influences the classification mechanism of the SVM [15]. Research has shown that if the RBF kernel is used with parameter selection, there is no need to consider the linear kernel [16] as the former clearly out-performs the latter. It has also been shown that the RBF kernel is generally a better choice than the sigmoid kernel [17]. The polynomial kernel may be comparable to the RBF kernel, but it is significantly more complex, with more hyper parameters and an overall complex final model. Therefore, the RBF is the best choice of kernel for most applications.

This choice is further confirmed by its very successful application in a variety of classification contexts [18, 19], in addition to hand shape recognition in a single orientation [20, 21]. Therefore, the RBF kernel will most likely provide classification performance that is at least as good as other kernels, but likely better.

5.3 Multi-Class SVM Classification Techniques

As observed in the SVM formulation, SVMs are intrinsically binary classifiers. Several techniques have been developed to apply SVMs to multi-class problems [22].

Most of these techniques combine several SVMs and use a decision strategy to select a single class. The following subsections describe three prominent techniques [22].

5.3.1 One-Versus-All

Given a K-class problem, the classification problem is to separate the points of each class k from the points of the other $(K - 1)$ classes, for every $k \in \{1, \dots, K\}$. Therefore, for each class k, a binary SVM is trained to separate the data points of class k from the data points of all classes other than class k. This results in a total of K binary classifiers.

Given an unknown data point that requires placement into one of the K classes, the pattern is presented to each of the K classifiers. The class of the pattern is determined to be the class that receives a vote from one of the K classifiers. This method has a relatively high training complexity given each classifier is effectively trained on all of the data.

5.3.2 One-Versus-One

Similar to the previous technique, this technique produces a series of SVMs. In this case, however, each SVM is trained to discern between two specific classes k and t, where $k \neq t$, for every distinct pair (k, t) where $k, t \in \{1, \dots, K\}$. The data points belonging to the relevant classes in each case are used to train the relevant SVM. This results in a total of $\frac{K(K-1)}{2}$ binary classifiers.

As with the previous technique, an unknown test pattern is presented to all the classifiers. In this case, the class of the pattern is determined to be the class that receives the largest number of votes across all the classifiers. Despite using a larger number of classifiers than the one-versus-all technique, this technique has a lower computational complexity in training, given each individual classifier is only trained on a small subset of the data. When classifying an unknown pattern, it may be considered to be more complex given the larger number of classifiers to query. In terms of classification accuracy, however, it has been shown that this technique is comparable to the one-versus-all technique [22, 23]. It is, therefore, preferred to the one-versus-all technique.

5.3.3 Directed Acyclic Graph Support Vector Machine

The directed acyclic graph (DAG) multi-class technique for SVMs was first proposed by Platt et al. [24]. The training phase of the DAG multi-class classification technique is carried out as per the one-versus-one technique, i.e., one SVM is trained for every distinct pair of classes, resulting in a total of $\frac{K(K-1)}{2}$ SVMs.

Thereafter, a rooted binary DAG with $\frac{K(K-1)}{2}$ internal nodes and K leaves is used to classify unknown test patterns into one of the K classes. Each node in the graph is a one-versus-one SVM of classes k and t. At each level, one class is rejected, after which none of the SVMs involving that class will be invoked. This significantly reduces the number of classifiers that are invoked to make a prediction to only $K-1$.

An illustration of this process is provided in Fig. 8 with a 4-class problem with classes $k \in \{1, 2, 3, 4\}$. Starting at the root node, classes 1 and 2 are compared. If class 1 is determined to be the correct class, this implies that class 2 was rejected. Therefore, it is resolved that classifiers involving class 2 will no longer be invoked.

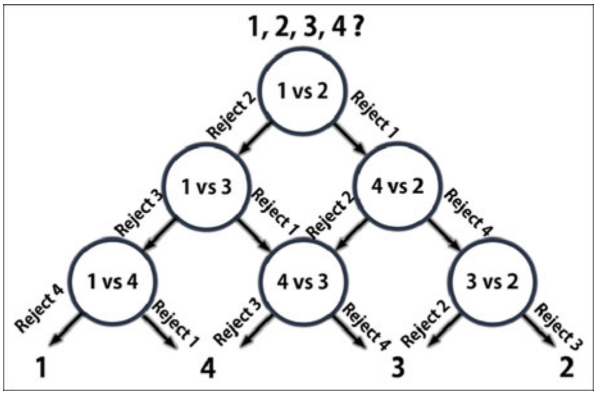


Fig. 8 A directed acyclic graph (DAG) of a 4-class problem

This concept is propagated down into all the remaining nodes. At each stage one class is rejected. At the end of the process, after only $K - 1$ steps, only a single non-rejected class remains at the bottom of the graph, which is taken to be the predicted class.

This technique combines the training efficiency of the one-versus-one technique with a classification efficiency that is better than the one-versus-all technique.

5.4 n-Fold Cross-Validation

If SVM is used for an n-class problem, then a validation technique is required that measures the correctness of all n classes. For this purpose, cross-validation is used. n-fold cross-validation is illustrated in Fig. 9. It involves dividing the training set into n unique subsets or "folds", represented by each of the vertical partitions of the data set in the figure. Then, n unique subsets are produced as shown in the figure. In each subset, one specific fold—shaded in the figure—is used for testing after having trained the classifier on all other folds—unshaded in the figure. The cross-validation accuracy for each specific parameter configuration is then the average accuracy over all n folds.

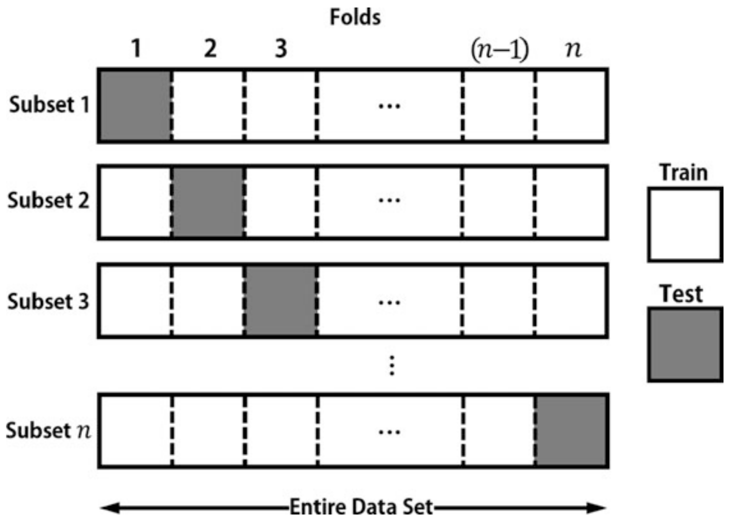


Fig. 9 Illustration of n-fold cross-validation

6 Conclusion

Several key techniques in computer vision and motion detection have been discussed in this work: Gaussian thresholding and mixture modeling for edge detection; cross correlation template matching for shape detection; the Viola–Jones model for face detection; and Gaussian mixture modeling for motion detection. For image classification, the HOG descriptor and SVM have been presented. Multi-class SVM classification techniques are elucidated, including one-versus-one, one-versus-all, and directed acyclic graph SVM. Cross-validation for these non-binary classifiers is also discussed.

Acknowledgment This research is supported in part by the National Research Foundation of South Africa (UNIQUE GRANT NO: 105670).

References

1. M. Ghaziasgar, Automatic sign language manual parameter recognition, Ph.D. thesis, University of the Western Cape, Computer Science, 2017
2. A.K. Jain, *Fundamentals of Digital Image Processing* (Prentice-Hall, Upper Saddle River, 1989)
3. G. Bradski, A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library* (O'Reilly Media, Sebastopol, 2008)
4. P. Viola, M.J. Jones, Robust real-time face detection. Int. J. Comput. Vis. **57**(2), 137–154 (2004)
5. M.J. Taylor, T. Morris, Adaptive skin segmentation via feature-based face detection, in *Proceedings SPIE Photonics Europe* (International Society for Optics and Photonics, Bellingham, 2014), pp. 91390P–1–12
6. R.J. Radke, S. Andra, O. Al-Kofahi, B. Roysam, Image change detection algorithms: a systematic survey. IEEE Trans. Image Process. **14**(3), 294–307 (2005)
7. H.S. Parekh, D.G. Thakore, U.K. Jaliya, A survey on object detection and tracking methods. Int. J. Innov. Res. Comput. Commun. Eng. **2**(2), 2970–2979 (2014)
8. Z. Zivkovic, Improved adaptive Gaussian mixture model for background subtraction, in *Proceedings 17th International Conference on Pattern Recognition*, vol. 2 (IEEE, Piscataway, 2004), pp. 28–31
9. R.K. McConnell, Method of and apparatus for pattern recognition, US Patent 4,567,610 (1986, 28 Jan)
10. N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1 (IEEE, Piscataway, 2005), pp. 886–893
11. Y. Yang, L. Lin, Automatic pedestrians segmentation based on machine learning in surveillance video, in *2019 IEEE International Conference on Computational Electromagnetics (ICCEM)* (IEEE, Piscataway, 2019), pp. 1–3
12. C. Wang, Z. Li, N. Dey, Z. Li, A.S. Ashour, S.J. Fong, R.S. Sherratt, L. Wu, F. Shi, Histogram of oriented gradient based plantar pressure image feature extraction and classification employing fuzzy support vector machine. J. Med. Imaging Health Informatics **8**(4), 842–854 (2018)
13. R. Kapoor, R. Gupta, S. Jha, R. Kumar et al., Detection of power quality event using histogram of oriented gradients and support vector machine. Measurement **120**, 52–75 (2018)
14. C. Cortes, V. Vapnik, Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)

15. S. Gupta, Support vector machines based modelling of concrete strength. World Acad. Sci. Eng. Technol. **36**, 305–311 (2007)
16. S.S. Keerthi, C.-J. Lin, Asymptotic behaviors of support vector machines with Gaussian kernel. Neural Comput. **15**(7), 1667–1689 (2003)
17. H.-T. Lin, C.-J. Lin, A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. Neural Comput. **3**, 1–32 (2003)
18. K.O. Rodriguez, G.C. Chavez, Finger spelling recognition from RGB-D information using kernel descriptor, in *Proceedings XXVI Conference on Graphics, Patterns and Images* (IEEE, Piscataway, 2013), pp. 1–7
19. I. Nitze, U. Schulthess, H. Asche, Comparison of machine learning algorithms random forest, artificial neural network and support vector machine to maximum likelihood for supervised crop type classification, in *Proceedings 4th GEOBIA* (2012), pp. 7–9
20. P. Li, M. Ghaziasgar, J. Connan, Hand shape recognition and estimation for South African sign language, in *Proceedings South African Telecommunication Networks and Applications Conference* (2011), pp. 344–349
21. R. Foster, M. Ghaziasgar, J. Connan, R. Dodds, A comparison of machine learning techniques for hand shape recognition, in *Proceedings South African Telecommunication Networks and Applications Conference, Port Elizabeth* (2014), pp. 173–178
22. C.-W. Hsu, C.-J. Lin, A comparison of methods for multiclass support vector machines. IEEE Trans. Neural Netw. **13**(2), 415–425 (2002)
23. K.-B. Duan, S.S. Keerthi, Which is the best multiclass SVM method? An empirical study, in *International Workshop on Multiple Classifier Systems* (Springer, Berlin, 2005), pp. 278–285
24. J.C. Platt, N. Cristianini, J. Shawe-Taylor, Large margin DAGs for multiclass classification, in *NIPS*, vol. 12 (1999), pp. 547–553

Overview of Deep Learning in Facial Recognition

Arnauld Fountsop Nzegha, Jean Louis Ebongue Fendji, Christopher Thron, and Clementin Djameni Tayou

1 Introduction

Deep learning is a growing trend in general data analysis, and it is generally considered to be a breakthrough technology in the last decade [30]. Deep learning relies on *artificial neural networks*, which are computational models composed of a number of layers of programmed "neurons." These models allow higher abstraction levels and better predictions from the data. They have met the need for faster and more efficient processing techniques required for burgeoning amounts of digital imaging data. Deep learning has enabled great progress in various applications of computer vision, such as image classification [9], object detection [16, 26], and facial recognition [11, 20, 33, 34, 39, 47, 53].

Facial recognition is usually divided into two subcategories: facial verification (or 1:1 face recognition), which checks whether a face corresponds to a given identity; and facial identification (or 1:N face recognition), which finds the identity matching a given face. Deep learning simulates the human nervous system's process of visual perception, which makes it possible to learn a deeper and more representative characterization of facial data.

A. F. Nzegha (✉) · C. D. Tayou
Department of Mathematics and Computer Science, University of Dschang, Dschang, Cameroon
e-mail: arnauldnzegha@gmail.com

J. L. E. Fendji
Department of Computer Engineering, Institute of Technology, University of Ngaoundere, Ngaoundere, Cameroon

C. Thron
Department of Science and Mathematics, Texas A&M University-Central Texas, Killeen, TX, USA

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_6

2 Neural Nets: Basic Structure and Function

2.1 History

The origins of deep learning date from 1943, with the introduction of the McCulloch–Pitts model (MCP) [36], which became the prototype of artificial neural models. McCulloch-Pitts et al. created a computer model based on neural networks that functionally mimics the neocortex in the human brain. The model used the so-called threshold logic to simulate the all-or-nothing firing behavior of neurons. However, they did not include learning in their model. This deficiency was addressed by Donald Hebb, who in 1949 formulated the “Hebbian theory” that explained learning in biological organisms in terms of changes in the firing behavior of neurons that have been repeatedly stimulated [24].

Inspired by the McCulloch–Pitts model, some researchers began to investigate the possibility of using neural networks to perform recognition tasks. The “perceptron,” a hardware device designed to perform simple image recognition tasks using a combination of 400 inputs from a 20 $\times$ 20 array of photocells was built in 1958 by Frank Rosenblatt at the Cornell Aeronautical Laboratory [41]. In 1974 Paul J. Werbos [56] introduced backpropagation, which makes use of a trial and error process to tune the parameters of neural networks models so as to improve their recognition capabilities. This opened the way to modern neural networks which can effectively learn how to classify data by making use of a training set.

Inspired by the work of Hubel and Wiesel [23] on the visual cortex in mammals, Kunihiko Fukushima utilized cascading architecture and concepts such as weight sharing in his so-called neocognitron [15], developed in 1983. The neocognitron can be considered as the ancestor of convolutional neural networks. However, it was not until the 1990s that convolutional neural networks were popularized by Yan LeCun and his team on the recognition of handwritten characters [29]. Due to the hardware limitations of the time, the structure of LeCun’s system (called LeNet-5) is fairly naive and could only work on very small-sized images of 28 by 28 black-and-white pixels. “Deep learning” refers to more complex systems that employ convolutional neural networks with multiple layers. Such deep learning systems set the current performance bar for facial recognition, and are capable of accurately separating identities based on complex facial image data.

2.2 Basic Concepts and Constructs in Deep Learning

2.2.1 Single-Layer Perceptron

In the human brain, each neuron is connected to an average of about ten thousand others through synapses located on dendrites and axon terminals (see Fig. 1). Each neuron receives electrical pulses of various intensities from other neurons: if the

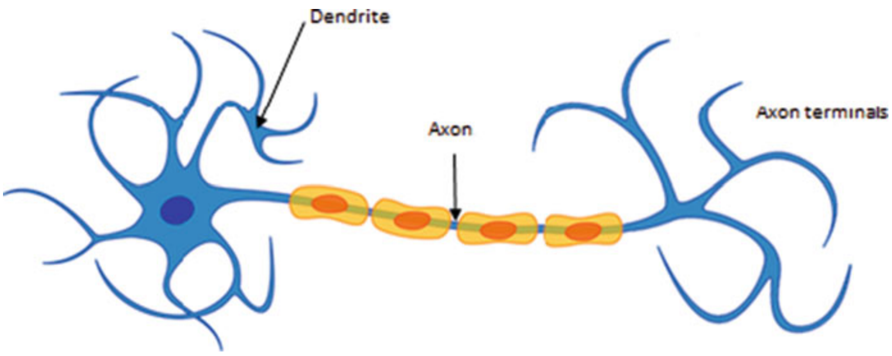


Fig. 1 Biological neuron, with dendrites and axons where synapses are located

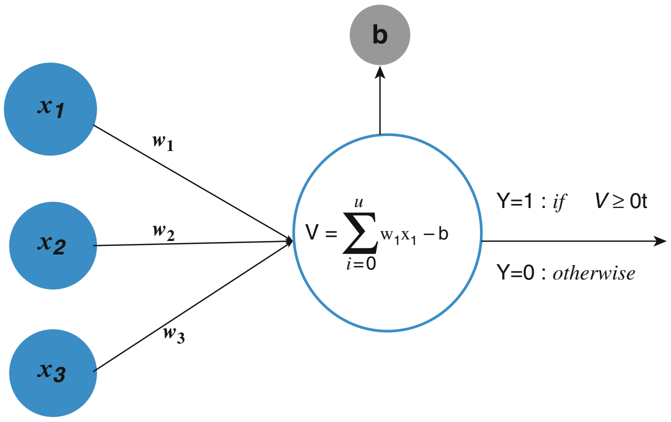


Fig. 2 Perceptron (mathematical neuron)

sum of the pulse intensities exceeds a certain threshold, then the neuron is activated, causing it to transmit an electrical pulse to connected neurons or to other cells [23]. The nerve's activation can be modeled as 0 or 1 (respectively, no activation and activation). The mathematical neuron model used in neural networks mimics the functioning of these biological neurons.

In modern terminology, a single mathematical neuron is called a *perceptron* (or more specifically a *single-layer perceptron*, or SLP). We will also refer to a SLP as a "unit." Figure 2 shows a schematic diagram of a perceptron. The formal specification of a SLP is as follows:

- A vector of inputs $X = [x_1, x_2, \ldots, x_N]$ of length N which models the inputs of the unit;
- A vector of weights $W = [w_1, w_2, \ldots, w_N]$, also of length N, which models all the synaptic weights of the unit. Each component w_i gives the weight of corresponding to the ith entry of X.

- A bias b that corresponds to the activation threshold of the unit.

The *potential* of the SLP (denoted by V) is calculated as the weighted sum of inputs (i.e., the inner product of the input vector with the weight vector), minus the bias:

$$V = \left(\sum_{i=1}^{N} w_i x_i \right) - b. \tag{1}$$

The potential is passed into an *activation function* (or *transfer function*) f which defines the response of the SLP. As discussed below, the choice of the activation function depends on the classification problem and the network model. Different possibilities for the activation function include:

- The *Heaviside function* corresponds to the 0–1 nerve response alluded to in Sect. 2.1:

$$H(V) = \{ {}^{0 \quad if \quad V \le 0}_{1 \quad otherwise}. \tag{2}$$

- The *ReLU* (rectified linear unit) *function* cuts off the negative part of the identity function, so that the SLP is only activates if the potential is positive. We shall write the ReLU function as $(\cdot)_+$, and the definition is as follows:

$$(V)_+ = \begin{cases} 0 & \text{if } V \le 0 \\ V & \text{if } V > 0 \end{cases} \tag{3}$$

or equivalently,

$$(V)_+ = \max(0, V). \tag{4}$$

As discussed below, this function is suitable for backpropagation because both the first derivative and second derivative have very simple expressions:

$$\frac{d}{dV}(V)_+ = H(V); \quad \frac{d^2}{dV^2}(V)_+ = 0 \text{ for all values of } V. \tag{5}$$

- The *logistic* (or *sigmoid*) *function*:

$$\sigma(V) = \frac{e^V}{(1 + e^V)}. \tag{6}$$

This function is not often used in propagation multilayer feed-forward neural networks because it "saturates," i.e., the output is nearly constant for large inputs. Nevertheless, many probabilistic models, recurrent networks, and some selfencoders have restrictions on outputs that make sigmoidal units attractive despite the disadvantages of saturation.

- The *tanh function*:

$$\tanh V = \frac{e^V - e^{-V}}{e^V + e^{-V}}. \tag{7}$$

The tanh and sigmoid functions are closely related: the reader may verify that $\tanh(x) = 2\sigma(2x) - 1$. The tanh function has the advantage that its derivative is 1 when $x = 0$, which facilitates training as we shall see later.

- *Radial basis functions*:

In the NN literature, radial basis functions (RBF) are mentioned frequently. Neurons that utilize RBFs have a different mathematical structure from SLP's. Instead of a potential that is linear in the inputs as in (1), the potential is related to the squared distance (i.e., radius) between the N-dimensional input vector and a point in N-dimensional space. Explicitly, the potential is given by

$$V = \frac{\|X - C\|^2}{2\sigma^2}, \tag{8}$$

where C is an N-dimensional vector, σ is a scaling factor, and the $\| \cdots \|^2$ notation represents the squared Euclidean distance, which is the sum of squared differences of all components:

$$\|X - C\|^2 := \sum_{n=1}^{N} (x_n - c_n)^2. \tag{9}$$

From (9) it may be seen that the center vector C and the scaling factor σ substitute for the weight vector W and bias b that are used in linear perceptrons.

Two radial basis functions that are often used in neural networks are:

– *Gaussian*: $\varphi(V) = e^{-V}$;
– *Multiquadric*: $\varphi(V) = \sqrt{1 + V}$.

In contrast to the activation functions in SLP's (which are increasing functions of the potential), RBFs are unimodal functions whose maximum is attained at 0, which according to (2.2.1) occurs when $X = C$. This reflects their use in identifying regions surrounding a particular point. In the following discussion we shall not make further use of RBFs—we have mentioned them here for completeness' sake.

Activation functions enable the unit to distinguish between classes of possible inputs. Consider the case where the unit has n inputs, so that the input may be considered as a N-dimensional vector or point in $\mathbb{R}^N$. Suppose there are two classes of inputs, denoted by C_1 and C_2, which we want the unit to distinguish. Mathematically, C_1 and C_2 may be represented as subsets of $\mathbb{R}^N$. We consider in

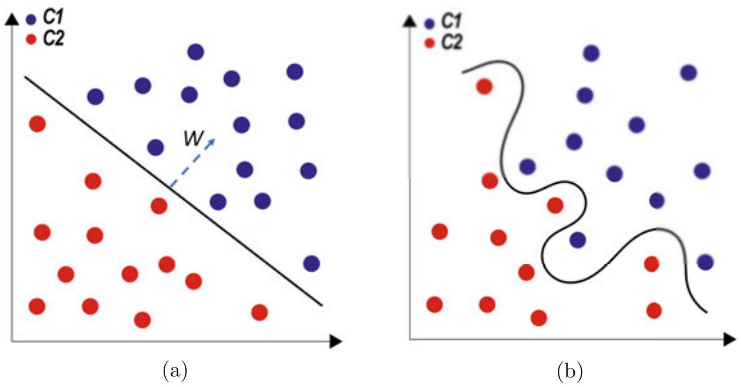


Fig. 3 Example of linearly and non-linearly separable classes. (**a**) Linearly separable classes. (**b**) Non-linearly separable classes

particular the case where C_1 and C_2 are *linearly separable*, which means that there exists a hyperplane $\mathcal{H}$ such that C_1 and C_2 both lie entirely on opposite sides of $\mathcal{H}$. An example of linearly separable classes in $\mathbb{R}^2$ is shown in Fig. 3a.

The hyperplane $\mathcal{H} \subset \mathbb{R}^N$ has a defining equation:

$$W \cdot X = b, \tag{10}$$

where W may be geometrically interpreted as a vector that is perpendicular to $\mathcal{H}$, and the scalar value b produces an offset so that $\mathcal{H}$ does not pass through the origin if $b \neq 0$.

If we let $V = W \cdot X - b$ as in (1), it follows that inputs belonging to C_1 and C_2 can be distinguished by the values of V which they produce

$$X \in C_1 \implies V > 0,$$
$$X \in C_2 \implies V < 0.$$

The activation function $f(V)$ conveys a degree of "confidence" for membership in C_1: The larger the value of $f(V)$ produced by X, the farther X is from $\mathcal{H}$ and the more "confident" we are that the point X is in C_1 and not in C_2.

2.2.2 The Multilayer Perceptron

A SLP can classify linearly separated data, but the data for many important applications is not linearly separable. For example, images can be represented as points in very large-dimensional spaces, and there may not exist a separating hyperplane to distinguish them. We see an example of non-linearly separable data in Fig. 3b.

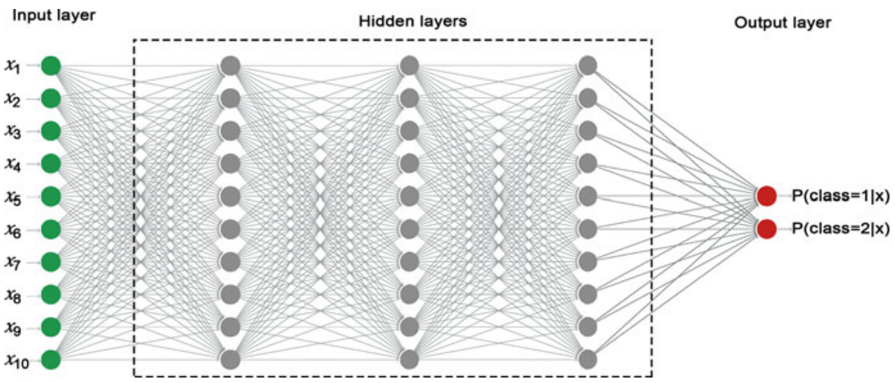


Fig. 4 Example of multilayer perceptron (MLP) with two hidden layers

The multilayer perceptron (MLP) is a classifier with a layered neural network structure. An MLP consists of three types of layers: the input layer, the hidden layers, and the output layer. As shown in Fig. 4, each layer of the MLP consists of multiple SLPs, which (unlike the perceptron in Fig. 2) may have multiple outputs. Each perceptron in the input layer has a single input, corresponding to a single component of the data vector: thus the number of units in the input layer is equal to the size of the data vector. Each unit in the network has its own output bias; and each connection between units in Fig. 4 has a different weight. The number of hidden layers and the number of units in each hidden layer can be chosen by the user: these numbers will typically depend on the complexity of the classification task. MLPs are not limited to linearly separable classification: it has been proven mathematically that an MLP can distinguish *any* two finite classes, no matter how complicated the boundary is that separates them [8]. Increasing the number of hidden layers may reduce the number of units and/or connections required for the classification.

2.2.3 Training of MLP's

Much of the usefulness of MLP's derives from the fact that they can be *trained* to perform classification tasks. Training involves a process whereby the MLP's parameters (biases and weights) are tuned with the aid of a set of training data. To explain the training process, we introduce some mathematical notation:

- Since training is an iterative process, we use t as an index to indicate the iteration number.
- M, N, J, and P denote, respectively, the number of layers, number of inputs, number of outputs, and number of vectors in the training set for the MLP;
- X is a P by N matrix, where each row of X is an input vector in the training set. The p'th row of X is denoted by X_p;

- $\widehat{Y}_t$ is a P by J matrix, where the p'th row of $\widehat{Y}_t$ is the *actual* output from the MLP corresponding to the input vector X_p (since the MLP parameters are changing from iteration to iteration, the output depends on t);
- $f_t^{(m)}$ is the vector function consisting of all activation functions at the m'th layer, $m = 1 \dots M$, at the t'th iteration. $f_t^{(1)}$ acts row by row on the rows of X, $f_t^{(2)}$ acts row by row on the rows of $f_t^{(1)}$, and so on. Note that $f_t^{(m)}$ has both vector inputs (the outputs from the previous layer) as well as vector outputs (to the next layer).

With the definitions above, we may express the MLP's output as a function of its input and activation functions at various layers as:

$$\widehat{Y}_t = f_t^{(M)} \circ f_t^{(M-1)} \circ \dots \circ f_t^{(1)}(X), \tag{11}$$

where "$\circ$" denotes the function composition and the functions $f_t^{(m)}$ are applied row by row to matrix inputs.

It is helpful to introduce the notation:

$$X^{(0)} := X; \quad X_t^{(m)} := f_t^{(m)}(X_t^{(m-1)}), \tag{12}$$

so that (11) becomes

$$\widehat{Y}_t = X_t^{(M)} = f_t^{(M)}(X_t^{(M-1)}). \tag{13}$$

Before we can describe the process of MLP training, we need a few additional definitions:

- $\Theta_t^{(m)}$ is the vector consisting of all MLP parameters at the m'th layer, including all input weights and all output biases for all neurons at that layer. Note that the vector function $f_t^{(m)}$ depends on the parameter vector $\Theta_t^{(m)}$ for layer m, but not on the parameter vectors for any other layers besides m. It follows that Eqs. (12)–(13) can be expressed more accurately as:

$$X_t^{(m)} := f^{(m)}(X_t^{(m-1)}, \Theta_t^{(m)}); \qquad \widehat{Y}_t = f^{(M)}(X_t^{(M-1)}, \Theta_t^{(M)}), \tag{14}$$

  but we will often use the notation in Eqs. (12)–(13) for short.
- Y is a J by P matrix, where the p'th row Y_p is the *desired* output corresponding to the input vector X_p;
- $\mathcal{L}(Y, \widehat{Y}_t)$ is the *loss function* (also called *cost function*, or *prediction error function*), which measures the difference between the desired and actual outputs. One commonly used loss function is *L2 loss* (also called *Euclidean loss*, which corresponds to the squared Euclidean distance between Y and $\widehat{Y}_t$ as defined in (9):

$$\mathcal{L}_{\mathrm{L2}}(Y, \widehat{Y}_t) := \|Y - \widehat{Y}_t\|^2. \tag{15}$$

Since Y and $\widehat{Y}_t$ are the matrices (as we have defined them), it follows that the loss function in (15) is the sum of squared entries of the matrix $Y - \widehat{Y}_t$. This loss function can be written as the sum of loss functions for each row:

$$\begin{aligned}\mathcal{L}_{\text{L2}}(Y, \widehat{Y}_t) &= \|Y - \widehat{Y}_t\|^2 && (16)\\ &= \|Y_1 - \widehat{Y}_{t,1}\|^2 + \|Y_2 - \widehat{Y}_{t,2}\|^2 + \ldots + \|Y_P - \widehat{Y}_{t,P}\|^2 && (17)\\ &= \mathcal{L}_{\text{L2}}(Y_1, \widehat{Y}_{t,1}) + \mathcal{L}_{\text{L2}}(Y_2, \widehat{Y}_{t,2}) + \ldots + \mathcal{L}_{\text{L2}}(Y_P, \widehat{Y}_{t,P}). && (18)\end{aligned}$$

It follows that the loss function for the entire training set can be computed as the sum of loss functions for the individual training vectors (some references use the term "loss function" to refer exclusively to the functions of individual vectors, while the term "cost function" is reserved for the sum of the loss functions).

A slight modification of the L2 loss is the *mean squared error loss* (or MSE loss) which is the average of the squared components of the error vector:

$$\mathcal{L}_{\text{MSE}}(Y, \widehat{Y}_t) := \frac{1}{J}\|Y - \widehat{Y}_t\|^2. \tag{19}$$

Since $\mathcal{L}_{\text{MSE}}$ and $\mathcal{L}_{\text{L2}}$ differ only by the constant factor $1/J$, there is no real difference between the two (the constant factor amounts to a rescaling of the learning rate (see below)). Other loss functions are described in Sect. 3.2.

The point of training is to adjust the parameter vectors $\Theta_t^{(m)}$ to reduce the loss function as much as possible. This can be done using the principle of *gradient descent*, which implies that the best way to reduce the loss function by adjusting parameters is to move the parameters towards the direction of the negative gradient (the "downhill" direction). This may be written as:

$$\Theta_{t+1}^{(m)} = \Theta_t^{(m)} - \mu g_t^{(m)}, \qquad \text{where } g_t^{(m)} := \nabla_{\Theta_t^{(m)}}\mathcal{L}(Y, \widehat{Y}_t), \quad m = 1 \ldots M. \tag{20}$$

The parameter $\mu < 1$ in (20) is called the *learning rate*, and it controls the rate at which the parameters adjust at each iteration. Typically, μ is chosen to be rather small so that the parameters do not "overreact." In Sect. 3.3, we will further discuss issues with the choice of μ.

Based on the above descriptions, the training process can be summarized as follows:

1. Compute $X_t^{(1)}, X_t^{(2)}, \ldots, X_t^{(M)}$ using (12)–(13);
2. Compute $g_t^{(m)}$ for $m = 1, \ldots, M$ (this computation will be described below);
3. Adjust the parameter vectors $\{\Theta_t^{(m)}\}_{m=1\ldots M}$ according to (20);
4. Iterate steps (1)–(3) until no further decrease in $\mathcal{L}(Y, \widehat{Y}_t)$ can be obtained.

The algorithm as described above is known at *batch propagation*. The reason for this terminology is that the matrices $X_t^{(1)}, \ldots, X_t^{(M)}$ are computed at each iteration

t, and each matrix corresponds to the whole "batch" of input vectors. There are alternative propagation algorithms that tend to exhibit faster convergence. Perhaps the most popular is *stochastic gradient descent*. In this variant, at each iteration computations are performed for a single row X_p instead of the entire matrix X. The algorithm is called "stochastic" because the sequence of rows used is randomized: the rows are cycled through multiple times, and before each new cycle the row order is randomly shuffled. Intermediate between ordinary and batch propagation is "minibatch" propagation, where the rows are divided into groups of rows, and the groups are cycled through and randomly shuffled following each cycle. There are other variants of this training algorithm that share the same general outline in which steps 1–4 are modified in different ways: see [44] for a review.

To complete our specification of the training process, we must show how to compute the gradients $g_t^{(m)}$ for $m = 1, \dots, M$. Without loss of generality we may consider the case of a single input vector (i.e., $P = 1$), which corresponds to stochastic gradient descent. To obtain the gradient for batch propagation, according to (16) we simply add the gradients for all the input vectors in the batch.

The basic mathematical tool used in computing the gradients $g_t^{(m)}$ in (20) is the *chain rule* from calculus, which gives a formula for the derivative of the composition of functions. Here we give several versions of the chain rule for reference. If a and b are the scalar functions and x is a scalar variable, we have

$$\frac{da}{dx} = \frac{da}{db}\frac{db}{dx}. \tag{21}$$

If a is a scalar function and b and x are the vectors, then we have

$$\nabla_x a = \nabla_b a \cdot \mathcal{J}_x b, \tag{22}$$

where $\nabla_x a$ and ∇b_a are the *gradient vectors* (written as row vectors):

$$\nabla_x a = \begin{bmatrix} \frac{\partial a}{\partial x_1} & \frac{\partial a}{\partial x_2} & \dots \end{bmatrix} \quad \text{and} \quad \nabla_b a = \begin{bmatrix} \frac{\partial a}{\partial b_1} & \frac{\partial a}{\partial b_2} & \dots \end{bmatrix}, \tag{23}$$

and $\mathcal{J}_x b$ is the *Jacobian matrix* of partial derivatives,

$$\mathcal{J}_x b = \begin{bmatrix} \frac{\partial b_1}{\partial x_1} & \frac{\partial b_2}{\partial x_1} & \dots \\ \frac{\partial b_1}{\partial x_2} & \frac{\partial b_2}{\partial x_2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{24}$$

(note the length of the vector $\nabla_x a$ is the same as the length of x, and the number of rows and columns of $\mathcal{J}_x b$ is equal to the lengths of x and b, respectively).

Using these equations, we may compute

$$\begin{aligned} g_t^{(M)} = \nabla_{\Theta_t^{(M)}} \mathcal{L}(Y, \widehat{Y}_t) &= \left(\nabla_{\widehat{Y}_t} \mathcal{L}\right) \cdot \left(\mathcal{J}_{\Theta_t^{(M)}} \widehat{Y}_t\right) \\ &= \left(\nabla_{\widehat{Y}_t} \mathcal{L}\right) \cdot \left(\mathcal{J}_{\Theta_t^{(M)}} f^{(M)}(X_t^{(M-1)}, \Theta_t^{(M)}) \right), \end{aligned} \tag{25}$$

where the gradient and Jacobian on the right-hand side of (25) may be computed directly from the functional forms of $\mathcal{L}$ and $f^{(M)}$, respectively (we will give an example later). Continuing to the previous layer, once again using the chain rule we have (writing $f^{(m)}(X_t^{(m-1)}, \Theta_t^{(m)})$ as $f_t^{(m)}$ for short):

$$\begin{aligned} g_t^{(M-1)} &= \left(\nabla_{\widehat{Y}_t} \mathcal{L}\right) \cdot \left(\mathcal{J}_{\Theta_t^{(M-1)}} \widehat{Y}_t\right) \\ &= \left(\nabla_{\widehat{Y}_t} \mathcal{L}\right) \cdot \left(\mathcal{J}_{\Theta_t^{(M-1)}} f_t^{(M)}\right) \\ &= \left(\nabla_{\widehat{Y}_t} \mathcal{L}\right) \cdot \left(\mathcal{J}_{X_t^{(M-1)}} f_t^{(M)}\right) \cdot \left(\mathcal{J}_{\Theta_t^{(M-1)}} X_t^{(M-1)}\right) \\ &= \left(\nabla_{\widehat{Y}_t} \mathcal{L}\right) \cdot \left(\mathcal{J}_{X_t^{(M-1)}} f_t^{(M)}\right) \cdot \left(\mathcal{J}_{\Theta_t^{(M-1)}} f_t^{(M-1)}\right), \end{aligned} \tag{26}$$

and once again all Jacobians may be computed directly (note that the first gradient in (26) was already computed in (25)). For the general m'th layer, by continuing the above process we have

$$\begin{aligned} g_t^{(m)} &= \left(\nabla_{\widehat{Y}_t} \mathcal{L}\right) \cdot \left(\mathcal{J}_{\Theta_t^{(m)}} \widehat{Y}_t\right) \\ &= \left(\nabla_{\widehat{Y}_t} \mathcal{L}\right) \cdot \left(\mathcal{J}_{\Theta_t^{(m)}} f_t^{(M)}\right) \\ &= \ldots \\ &= \left(\nabla_{\widehat{Y}_t} \mathcal{L}\right) \cdot \left(\mathcal{J}_{X_t^{(M-1)}} f_t^{(M)}\right) \cdot \ldots \cdot \left(\mathcal{J}_{X_t^{(m)}} f_t^{(m+1)}\right) \cdot \left(\mathcal{J}_{\Theta_t^{(m)}} f_t^{(m)}\right), \end{aligned} \tag{27}$$

where all but the last two Jacobians that appear in (27) are reused from the computation for $g_t^{(m+1)}$. Once all of the gradients $g_t^{(m)}$ for $m = 1 \ldots M$ have been computed, the parameters $\{\Theta_t^{(m)}\}$ are adjusted according to (20), and the process is repeated.

We close this section with a particular example of the gradient calculation summarized in (27). Suppose that the L2 loss function is used, and all activation functions are ReLU functions. Then we may define the parameter vector at the mth layer as:

$$\Theta_t^{(m)} := [\theta_{t,1}^{(m)}, \ldots \theta_{t,N_m}^{(m)}], \tag{28}$$

where N_m is the number of neurons in the mth layer and

$$\theta_{t,n}^{(m)} := [W_{t,n}^{(m)}, b_{t,n}^{(m)}], \tag{29}$$

where $W_{t,n}^{(m)}$ and $b_{t,n}^{(m)}$ are, respectively, the weight vector and bias for the nth neuron in the mth layer at the tth iteration.

Now, the function $f_t^{(m)}$ is actually a vector of N_m activation functions expressing the outputs of the N_m neurons in the mth layer. So we may write

$$f_t^{(m)} = \left[\; (V_{t,1}^{(m)})_+ \;\; (V_{t,2}^{(m)})_+ \;\; \cdots \;\; (V_{t,N_m}^{(m)})_+ \; \right], \tag{30}$$

where

$$V_{t,n}^{(m)} = W_{t,n}^{(m)} \cdot X_t^{(m-1)} - b_{n,t}^{(m)}. \tag{31}$$

Note that $V_{t,n}^{(m)}$ depends only on $\Theta_t^{(m)}$ and not on $\theta_t^{(k)}$ for $k \neq m$. It follows that $\mathcal{J}_{\Theta_t^{(m)}} f_t^{(m)}$ can be written as:

$$\mathcal{J}_{\Theta_t^{(m)}} f_t^{(m)} = \begin{bmatrix} \nabla_{\theta_{t,1}^{(m)}} (V_{t,1}^{(m)})_+ \\ \nabla_{\theta_{t,2}^{(m)}} (V_{t,2}^{(m)})_+ \\ \vdots \\ \nabla_{\theta_{t,N_m}^{(m)}} (V_{t,N_m}^{(m)})_+ \end{bmatrix} = \begin{bmatrix} H(V_{t,1}^{(m)}) \\ H(V_{t,2}^{(m)}) \\ \vdots \\ H(V_{t,N_m}^{(m)}) \end{bmatrix} \left[X_t^{(m-1)}, -1 \right], \tag{32}$$

where we have used expression (5) for the derivative of the ReLU function. Similarly,

$$\mathcal{J}_{X_t^{(m-1)}} f_t^{(m)} = \begin{bmatrix} \nabla_{\theta_{t,1}^{(m)}} (V_{t,1}^{(m)})_+ \\ \nabla_{\theta_{t,2}^{(m)}} (V_{t,2}^{(m)})_+ \\ \vdots \\ \nabla_{\theta_{t,N_m}^{(m)}} (V_{t,N_m}^{(m)})_+ \end{bmatrix} = \begin{bmatrix} H(V_{t,1}^{(m)}) \\ H(V_{t,2}^{(m)}) \\ \vdots \\ H(V_{t,N_m}^{(m)}) \end{bmatrix} \left[X_t^{(m-1)}, -1 \right]. \tag{33}$$

Finally, the gradient $\nabla_{\widehat{Y}_t} \mathcal{L}(Y, \widehat{Y}_t)$ may be computed as

$$\nabla_{\widehat{Y}_t} \mathcal{L}(Y, \widehat{Y}_t) = \nabla_{\widehat{Y}_t} \|Y - \widehat{Y}_t\|^2 = -2(Y - \widehat{Y}_t). \tag{34}$$

2.3 Underlearning and Overlearning

The purpose of training a model is to learn the characteristics from the examples of a given dataset and to generalize them into other data in the domain. If the training is not done properly, either underlearning or overlearning may result, both of which lead to poor model performance. In this section we discuss these two phenomena.

Underlearning occurs when a classification model is unable to correctly classify the examples of the dataset on which it was trained. This is reflected in a high value of the prediction error at the end of the training process. Underlearning may be due to undertraining: the model has not been sufficiently trained to handle all the descriptors necessary for classification. Alternatively, underlearning occurs when the model used is too simple to describe the relationship between the input data and the target. In the case of neural networks, the classification capability is tied to model characteristics such number of layers, numbers of units per layer, number of weights, and so on (these characteristics are known as *hyperparameters*).

Undertraining can be distinguished from model oversimplification by examining the values of prediction error over the course of the training process. If the prediction error plateaus before the end of the process, then this is a sign that the model is too simple and the hyperparameters should be readjusted.

In comparison to underlearning, overlearning (or overfitting) is more difficult to detect and correct. Overlearning refers to the case where noise or random fluctuations in training data are perceived by the model as part of the classification. This may occur when the training set is either too small or not representative. For example, if a training set of cat photos includes only photos of striped or spotted cats, then the model may perceive stripedness or spottedness as a necessary aspect of “catness.” Later in this chapter we will address some strategies that can be used to combat overlearning.

2.4 Convolutional Neural Networks (CNN)

One type of neural network is the convolutional neural network (CNN) is a type of neural network that is particularly suited for image data (or more generally, any data which can be naturally arranged in a two-dimensional grid). In CNNs, the units in each layer are grouped as filters that respond to patches the visual field, as shown in Fig. 5. A bank of filters may cover the same patch, such that each filter in the bank extracts different information. This bank structure is replicated so that the set of all banks together constitute a “sliding window” that covers the entire image with overlapping patches (these patches are known as *receptive fields*). In addition, outputs from different patches may be combined in the next layer, so that CNNs are effectively able to scan the image at multiple resolutions in order to pick up both fine detail and larger-scale features. More information about how this is done is given in subsequent sections.

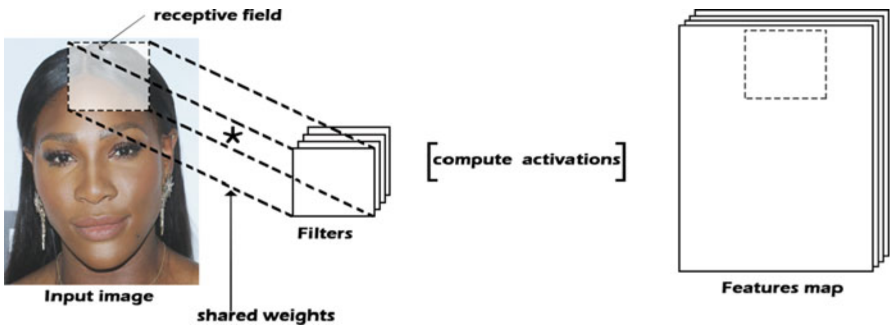


Fig. 5 Convolutions of an input image with a bank of four filters, yielding four feature maps

2.4.1 Convolutional Layers

The CNN has a layered structure similar to the model MLP shown in Fig. 4. The hidden layers of the CNN differ from those in Fig. 4 in that units in successive are not fully interconnected, but only "patchwise." Each filter shown in Fig. 5 corresponds to a matrix of weights which serves as a "window" that produces a single output based from the inputs of a single patch (or receptive field). Sliding each window around the entire image produces *convolutions* of the original image (one convolution for each filter) to which the activation function is applied pointwise, yielding *feature maps* as shown in Fig. 5.

2.4.2 Guiding Principles of Convolutional Layer Design

The description of CNNs above indicates three key principles that motivate the CNN layer design:

- Local connectivity: Images have a grid-like structure and each pixel is more dependent on its neighbors than on distant pixels. The disadvantage of using fully connected layers is that such a network architecture does not take into account the spatial structure of the data. Following the concept of receptive fields, CNNs exploit spatially local correlation by enforcing a local connectivity pattern between units of adjacent layers. Connections are local in space (along width and height), and evolve to global processing in deeper layers.
- Parsimonious connectivity: Local connectivity is implemented by means of weight matrices, which are small compared to the image size. The weight matrix pattern is repeated as a sliding window that scans the entire region, one receptive field at a time. This leads to a greatly reduced number of connections, compared to the full connectivity that we saw in MLPs. For example, given an input image of size (200×200), a CNN could employ a basic weight matrix of size (3×3), leading to $200 \times 200 \times 3 \times 3 = 360{,}000$ connections if the scanning window is

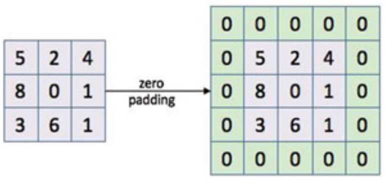
applied to every 3×3 patch in the image. This is far fewer connections than if the layers were fully connected, which would lead to $200^4 = 1.6$ billion connections.

- Weight sharing: Since the same weights are used for each receptive field, it is only necessary to store the weights for a single receptive field. This greatly reduces the number of weights that must be adjusted in the training process, leading to faster convergence. The fact that the weights are reused means that if a filter is capable of detecting a particular pattern, it can be detected independently of the receptive field where the pattern occurs: this property is known as *translation invariance*.

2.4.3 CNN Layer Hyperparameters: Window Size, Depth, Stride, and Padding

Each layer in the CNN is characterized by the following four hyperparameters:

- We have explained *window size* (or reception field size) and its importance above. The window size is determined by the size of characteristic features that are expected to be found in the image.
- The *depth* of the layer is the number of filters that operate in parallel on each window within the image. In the case of Fig. 5, the depth is four, and the four filters produce four feature maps (convolution images) of the original image.
- The *stride* is the distance between successive windows that are implemented in the network. Recall that the layer consists of multiple overlapping windows that cover the entire image: the stride controls the overlap of the receptive fields. The smaller the stride, the more overlapping there is between receptive fields and the more outputs produced by the layer. A stride of 1 means that successive windows are displaced by a single pixel; a stride of 2 means that there are windows centered on every other pixel; and so on.
- *Padding* is a technique for managing the edges of the input matrix. Figure 6 gives an example of padding, where a 3×3 input matrix is padded by a border of zeros. When windows of size 3×3 with a stride of 1 are applied to the padded image, the output matrix for each is also of size 3×3. Without padding, since there is only one 3×3 window the matrix of outputs is restricted to size 1×1. Thus padding (and zero padding in particular, which is the most common form of padding) can be used to ensure that the output matrix size is the same as the input matrix size. This prevents the loss of feature information near the boundary.



$$\begin{bmatrix} 5 & 2 & 4 \\ 8 & 0 & 1 \\ 3 & 6 & 1 \end{bmatrix} \xrightarrow{\text{zero padding}} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 2 & 4 & 0 \\ 0 & 8 & 0 & 1 & 0 \\ 0 & 3 & 6 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 6 Zero padding of a 3×3 image matrix

2.4.4 Pooling

We have seen above that each CNN layer produces multiple feature maps from the input to the layer, where the number of feature maps is equal to the depth of the layer. This implies that multiple successive layers will produce a multiplier effect, leading to larger and larger amounts of output data from the layers. In order to counteract this tendency, between operations the feature maps' sizes are reduced by local sampling or averaging. These size-reducing operations are known as *pooling*, and are accomplished by *pooling layers* inserted between the convolutional layers. Pooling operations are accomplished by dividing the pooling layer into rectangles, and performing the sampling or averaging on each rectangle.

There are several pooling techniques. Some of the most common are listed below:

- *Average pooling* consists of taking the average of all values in each rectangle. An example is shown in Fig. 7, where a 4 × 4 input to the pooling layer is reduced to a 2 × 2 output by taking averages over 2 × 2 squares.
- *Sum pooling* takes the sum on rectangles instead of the average. It is essentially the same as average pooling, differing only by a multiplicative factor;
- *Max-pooling* reduces each rectangle of pixels to the maximum pixel value. An example is shown in Fig. 8.

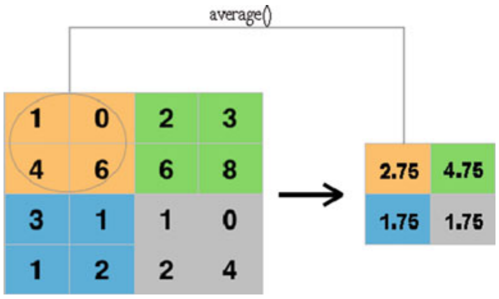


Fig. 7 Illustration of average-pooling operation

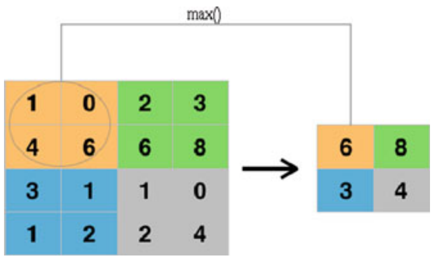


Fig. 8 Illustration of max-pooling operation

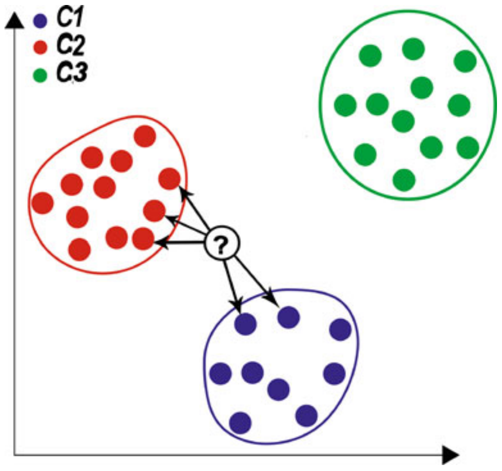


Fig. 9 Illustration of classification by the *k*-NN algorithm

2.4.5 Classifiers on CNN Outputs

After feature extraction by a CNN, it is possible to use another classifier such as the support vector machine (SVM), multilayer perceptron (MLP), or *k*-nearest neighbors (*k*NN) on the CNN outputs. These three classifiers are briefly described as follows.

k-nearest neighbors (*k*NN) is a simple machine learning algorithm that categorizes a data vector based on its proximity to other (preclassified) training data vectors in the same vector space [2, 54]. The concept is illustrated in Fig. 9. The algorithm requires that there be a “distance function” that measures the difference between two data vectors. (Euclidean distance may be used, if the data vectors are vectors in N-dimensional space: other distances that are sometimes used include Manhattan distance or Hamming distance, if the data are categorical.) Using the distance function, distances between the new vector and training vectors are computed, and the k training vectors with the shortest distances are identified (k is a hyperparameter chosen by the user). The new vector is then classified according to majority vote among the k nearest neighbors.

Support vector machine (SVM) methods are a family of supervised classifiers [6, 12]. SVM resembles the original perceptron in that if the training data is linearly separable, it produces a hyperplane that separates the training data into appropriate classes with the widest possible margin (see Fig. 10). This is done using standard numerical optimization techniques, not through iterative training as with MLPs. The optimization problem is mathematically recast in such a way that the solution depends only on inner products between training vectors, and the classification of a new vector depends only on the inner product between the new vector and the training vectors. In the case where the data is not linearly separated, the inner products in the optimization problem are replaced with a *kernel function*, which introduces a nonlinear transformation into the data which enables separation [28].

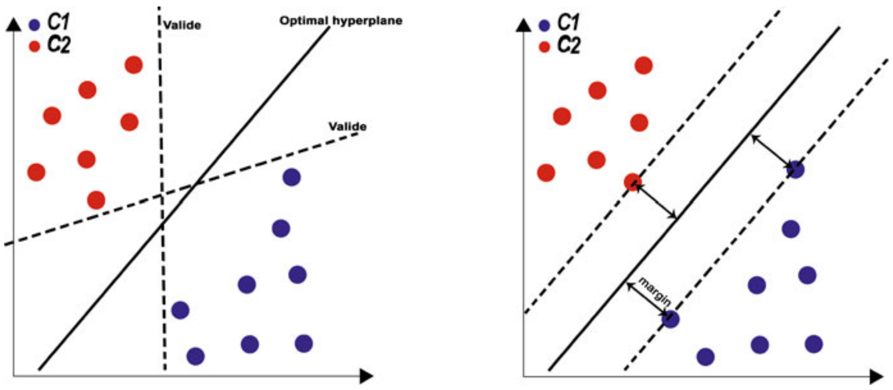


Fig. 10 SVM separating hyperplane for linearly separable data

If multiple classes are involved, then SVM can employ either *one-against all* classification (in which hyperplanes are found that separate each class from all the rest) or *pairwise* classification (in which separating hyperplanes are found for each pair of classes).

3 Neural Net Enhancements and Optimizations

3.1 Producing Probability Outputs with Softmax

If the MLP is used for classification, it may be useful to interpret the outputs as the probabilities of the different classes. For this to work, the outputs must be nonnegative and must sum to one, because these are basic properties that probabilities must satisfy. However, the outputs of MLPs we have discussed so far will in general satisfy neither of these two properties. It is necessary therefore to find a way to translate MLP outputs into probabilities. Hence we need a multidimensional function that acts on J real inputs and produces J nonnegative outputs that sum to 1. We may create a *multidimensional sigmoid* by turning real numbers to positive numbers through exponentiation (which preserves order) and multiplying by a common denominator to make the sum equal to 1:

$$\sigma(Y) = [\sigma_1(Y), \ldots, \sigma_J(Y)], \tag{35}$$

where

$$\sigma_i(Y) := \frac{e^{Y_i}}{\sum_{j=1}^{J} e^{Y_j}}. \tag{36}$$

If the vector Y has one component Y_k that is somewhat larger than the other components, then the k'th component of $\sigma(Y)$ will be close to 1 and the other components of $\sigma(Y)$ will be close to 0 (this is due to the exponential factor in the numerator in Eq. (35)). For this reason, the function $\sigma(Y)$ is called the *softmax* function. We will need later the partial derivatives of the softmax function, which may be computed as:

$$\frac{\partial \sigma_i(Y)}{\partial Y_j} = \begin{cases} \sigma_i(Y)(1-\sigma_j(Y)) \ if \ \ i=j \\ \quad -\sigma_i(Y)\sigma_j(Y) \quad if \ i \neq j. \end{cases} \tag{37}$$

This implies that the Jacobian (i.e., the $J \times J$ matrix of partial derivatives) $\mathcal{J}_Y(\sigma(Y))$ is given by

$$\mathcal{J}_Y(\sigma(Y)) = \text{diag}(\sigma(Y)) - \sigma(Y)^T\sigma(Y), \tag{38}$$

where diag $(\sigma(Y))$ is the $J \times J$ matrix whose diagonal entries are given by the vector $\sigma(Y)$ and the "T" superscript denotes the matrix transpose.

3.2 Loss Functions

We have already explained the importance of loss functions in Sect. 2.2.3. We also described the L2 and MSE loss functions (which yield exactly the same results, if the learning rate is rescaled by a constant factor). In this section we overview some other common loss functions and their properties. Many of these loss functions are used with CNNs for tasks such as facial recognition, but in theory they could be used with any MLP.

3.2.1 Cross-Entropy Loss

We saw in Sect. 3.1 that the softmax function can be used to convert MLP outputs into probabilities. In this case, directly applying L2 loss to the softmax output creates serious problems for convergence of the MLP. We may show this as follows. In (27) we showed that the MLP parameter adjustments are all proportional to the gradient of the loss function $\nabla_{\widehat{Y}}\mathcal{L}(Y,\widehat{Y})$. We denote the target probability distribution as the column vector T (usually T is an indicator vector that has a single entry 1 corresponding the known class of the input, and zeros in all other entries). Then we may evaluate the gradient of the L2 loss as follows (writing $\sigma(\widehat{Y})$ as σ):

$$\begin{aligned} \nabla_{\widehat{Y}}\|T-\sigma\|^2 &= \nabla_{\widehat{Y}}\left((T-\sigma)\cdot(T-\sigma)\right) \\ &= -2(T-\sigma)\left(\mathcal{J}_{\widehat{Y}}(\sigma)\right) \\ &= -2(T-\sigma)(\text{diag}(\sigma)-\sigma^T\sigma), \end{aligned} \tag{39}$$

where we have used the chain rule and (38) to obtain the second line and last line, respectively. If any of the components of $\sigma(\widehat{Y})$ is close to 0, then the corresponding component of the gradient is also very small, meaning that MLP is not very responsive to errors in that component. Accordingly, it is better to find a loss function whose gradient with respect to the MLP outputs is reasonable.

The loss function should measure the difference between probability distributions, because that is what our outputs are. One candidate for the such a loss function is the *Kullback–Leibler divergence* (or *relative entropy*), defined as follows. If $T = [T_1, \ldots, T_J]$ is the vector of target probabilities and $\sigma = \sigma(\hat{Y})$ is the probability distribution generated by the MLP, then the Kullback–Leibler divergence of σ with respect to T is $D_{KL}(\sigma||T)$, where

$$D_{KL}(\sigma||T) := T \cdot \log(T) - T \cdot \log(\sigma), \tag{40}$$

where "·" denotes the vector dot product. The quantity $D_{KL}(\sigma||T)$ is always nonnegative, and is equal to 0 if and only if $\sigma = T$. If σ closely resembles T, it will have small values of $D_{KL}(\sigma||T)$, and the value increases the more σ differs from T.

In our training process we are only interested in the derivatives of the loss function with respect to the MLP outputs $\widehat{Y}$. Since the first term on the left-hand side of (40) does not depend on $\sigma(\widehat{Y})$, we may remove it and use only the second term, which is called the *cross entropy*. We shall use $\mathcal{L}_{XE}$ to denote the cross-entropy loss function:

$$\mathcal{L}_{XE}(T, \sigma) := -T \cdot \log(\sigma), \tag{41}$$

and we compute

$$\nabla_\sigma \mathcal{L}_{XE}(T, \sigma) = -\left[\frac{T_1}{\sigma_1}, \ldots, \frac{T_J}{\sigma_J}\right] = -T \oslash \sigma, \tag{42}$$

where $\oslash$ denotes the componentwise division. From this it follows (again writing $\sigma(\widehat{Y})$ as σ):

$$\begin{aligned}\nabla_{\widehat{Y}} \mathcal{L}_{XE}(T, \sigma) &= -(T \oslash \sigma)\mathcal{J}_{\widehat{Y}}(\sigma)\\ &= -(T \oslash \sigma)(\operatorname{diag}(\sigma) - \sigma^T \sigma)\\ &= -TI + (T \cdot \mathbf{1})\sigma\\ &= -T + \sigma,\end{aligned} \tag{43}$$

where I is the identity matrix and $\mathbf{1}$ is the vector of all 1's (note $T \cdot \mathbf{1} = 1$ since T is a probability vector). Expression (43) is identical to $\nabla_\sigma \left(\|T - \sigma\|^2\right)$ (apart from a factor of 2): so we conclude that using the cross entropy as loss function on softmaxed outputs has the same effect as if the softmaxed outputs were direct outputs and the $L2$ norm was used.

In later discussions we will need a more detailed expression for the cross-entropy loss (41) in terms of the MLP input–output pair $(X, \widehat{Y})$. Let us introduce the notation $\mathcal{I}(X)$ to stand for the class that input vector X belongs to. We also suppose that the target vector T for the input vector X is the indicator vector for the index $\mathcal{I}(X)$:

$$T_i = \begin{cases} 1 & \text{if } i = \mathcal{I}(X) \\ 0 & \text{if } i \neq \mathcal{I}(X). \end{cases} \tag{44}$$

With these notations, we may rewrite (41) in terms of X and $\widehat{Y}$ as

$$\mathcal{L}_{XE}(X, \widehat{Y}) = -T \cdot \log(\sigma) = -\log\left(\frac{e^{\widehat{Y}_{\mathcal{I}(X)}}}{\sum_{j=1}^{J} e^{\widehat{Y}_j}}\right) = -\widehat{Y}_{\mathcal{I}(X)} + \log\left(\sum_{j=1}^{J} e^{\widehat{Y}_j}\right). \tag{45}$$

Let us suppose in addition that the J entries of the MLP output $\widehat{Y}$ are obtained via linear transformation of the output vector $X^{(M-1)}$ from the final hidden layer (i.e., there are no biases in the final layer). In this case we may write:

$$\widehat{Y} = X^{(M-1)}(W^{(M)})^T, \tag{46}$$

where $W^{(M)}$ is the matrix of input weights for the output layer. We may write this component-by-component as

$$\widehat{Y}_i = W_i^{(M)} \cdot X^{(M-1)}, \tag{47}$$

where $W_i^{(M)}$ is the i'th row of $W^{(M)}$. Using these notations, we may rewrite (45) as

$$\mathcal{L}_{XE}(X, X^{(M-1)}) = -W_{\mathcal{I}(X)}^{(M)} \cdot X^{(M-1)} + \log\left(\sum_{j=1}^{J} e^{W_j^{(M)} \cdot X^{(M-1)}}\right). \tag{48}$$

3.2.2 Contrastive Loss

So far we have described training as a process where MLP weights are adjusted so that the MLP outputs are close to a predetermined set of desired outputs. But in some image processing applications, this may not be the goal of the classification. It may be that the user simply wants to classify images according to similarity, and not assign them to predetermined classes. This means that the weights should be adjusted so that similar points are close in output space, and dissimilar points are far from each other.

Hadsell et al. in [19] compare this situation to a mechanical system where MLP outputs are point masses connected to each other by a series of springs. If the point masses represent similar images, the spring joining the points should pull them together; and if the images are dissimilar, the spring should push them apart. This mechanical system can be modeled mathematically by using a loss function that acts on *pairs* of input vectors, instead of individual input vectors as in the loss functions that we have previously examined. The loss function should model the potential energy of the spring, so that the negative gradient of the loss function (which guides the weight adjustment) models the force that pushes the points in the right direction.

Based on these specifications, we may define a pairwise loss function $\mathcal{L}_{pair}((X_1, \widehat{Y}_1)(X_2, \widehat{Y}_2))$ for the two input–output pairs $(X, \widehat{Y})$, $(X', \widehat{Y}')$ as:

$$\mathcal{L}_{pair}(X, \widehat{Y}, X', \widehat{Y}') = \begin{cases} \|\widehat{Y} - \widehat{Y}'\|^2 & \text{if } X \text{ and } X' \text{ are similar} \\ \left(m - \|\widehat{Y} - \widehat{Y}'\|^2\right)_+ & \text{if } X \text{ and } X' \text{ are dissimilar} \end{cases}, \tag{49}$$

where $\mathcal{L}_2$ is the L2 loss function defined in (15), the "+" subscript denotes the ReLU function, and m is a hyperparameter called the *margin*. In practice, the loss functions for P pairs (where P is a hyperparameter) are added together to form the contrastive loss function $\mathcal{L}_{CP}$:

$$\mathcal{L}_{CP}\left((X_1, \widehat{Y}_1, X'_1, \widehat{Y}'_1), \ldots, (X_P, \widehat{Y}_P, X'_P, \widehat{Y}'_P)\right) = \sum_{p=1}^{P} \mathcal{L}_{pair}(X_p, \widehat{Y}_p, X'_p, \widehat{Y}'_p). \tag{50}$$

3.2.3 Center Loss and Contrastive Center Loss

When softmax is used for classification, MLP outputs that lie in real J-dimensional space are transformed into probability vectors. During training, training vectors are inputted to the network, and the outputs are softmaxed and used to compute the loss function, which measures the difference between the softmaxed output and the desired probability vector. This process steers the outputs closer and closer to the desired output. However, it does not include any mechanism to make outputs belonging to different classes to be farther away from each other. One way to increase the separation between different classes is to make the outputs for each class closer to each other (and not just closer to the target). This can be done by including a term in the loss function that draws same-class outputs closer to each other. This is the motivation for the *center loss function* [55].

The center loss function is computed as follows. Initially, a center vector C_i is computed for each class i, $i = 1, \ldots, I$ as follows. Letting $\mathcal{S}_i$ denote the set of all training vectors that belong to the i'th class, we compute the class center vector by taking the average of outputs from the vectors in $\mathcal{S}_i$:

$$C_i = \frac{1}{|\mathcal{S}_i|} \sum_{X \in \mathcal{S}_i} \widehat{Y}(X). \tag{51}$$

Following this initialization, the training proceeds by minibatch backpropagation. Let $\mathcal{M}$ be the set of vectors in the minibatch, and let $\mathcal{M}_i$ be the set of training vectors in the minibatch that belong to class i. The center loss function for the minibatch is denoted by $\mathcal{L}_C$, and is defined as:

$$\mathcal{L}_C(\mathcal{M}) := \frac{1}{2} \sum_{i=1}^{I} \sum_{X \in \mathcal{M}_i} \|\widehat{Y}(X) - C_i\|^2. \tag{52}$$

$\mathcal{L}_C(\mathcal{M})$ penalizes vectors in $\mathcal{M}$ that are far from the center of the class they belong to. The MLP weights are then updated using (20). The center estimates are then updated based on the updated weights as follows:

$$(C_i)_{updated} = C_i + \frac{\sum_{X \in \mathcal{M}_i} (C_i - \widehat{Y}(X))}{|\mathcal{M}_i| + 1}. \tag{53}$$

In the implementation in [55], the loss function was a combination of center loss and cross-entropy loss:

$$\mathcal{L} = \sum_{i=1}^{I} \sum_{X \in \mathcal{M}_i} \mathcal{L}_{XE}\left(T_i, \sigma(\widehat{Y}(X))\right) + \lambda \mathcal{L}_C(\mathcal{M}), \tag{54}$$

where T_i is the target vector for the i'th class, and λ is a hyperparameter that controls the relative influence of the two loss functions.

Center loss is designed to bring output vectors in the same class closer together (in technical language: center loss reduces the intra-class variance). On the other hand, contrastive loss pushes output vectors in different classes farther apart (i.e., contrastive loss increases inter-class variance). It is possible to modify center loss in order to also increase inter-class variance. The *contrastive center loss function* is defined in [40] as follows:

$$\mathcal{L}_{CT-C}(\mathcal{M}) := \frac{1}{2} \sum_{i=1}^{I} \sum_{X \in \mathcal{M}_i} \frac{\|\widehat{Y}(X) - C_i\|^2}{\sum_{k \neq i} \|\widehat{Y}(X) - C_k\|^2 + 1}. \tag{55}$$

3.2.4 Triplet Loss

Contrastive loss (both without and with center loss) tries to move output points in the same class as close together as possible. But classification does not require that same-class outputs need to be so tightly clustered—rather, classification only requires that outputs from different classes are separated by a sufficient margin.

We may examine this requirement as it applies to three points X_a, X_p, X_n, where the first two points are in the same class and the third point is in a different class. If there is a margin m between the two classes, then the following inequality must hold [43]:

$$\|\widehat{Y}(X_a) - \widehat{Y}(X_n)\|^2 \geq m + \|\widehat{Y}(X_a) - \widehat{Y}(X_p)\|^2, \tag{56}$$

which may be rewritten as

$$m + \|\widehat{Y}(X_a) - \widehat{Y}(X_p)\|^2 - \|\widehat{Y}(X_a) - \widehat{Y}(X_n)\|^2 \leq 0. \tag{57}$$

To guarantee a margin m for all classes, it follows that similar inequalities must hold for *all* triplets where two are in the same class, and the third is in another class. For convenience, we introduce the notation $\{X\}$ to represent the class that input vector X belongs to. So the inequality (57) must hold for *every* X_p in $\{X_a\}$ and for *every* X_n that is not in $\{X_a\}$. The resulting inequalities will all be satisfied if and only if the following "worst case" inequality is satisfied:

$$m + \max_{X_p \in \{X_a\}} \|\widehat{Y}(X_a) - \widehat{Y}(X_p)\|^2 - \min_{X_n \notin \{X_a\}} \|\widehat{Y}(X_a) - \widehat{Y}(X_n)\|^2 \leq 0. \tag{58}$$

This suggests defining a *triplet loss function* $\mathcal{L}_{trip}$ as follows:

$$\mathcal{L}_{trip}(X_a) = \left(m + \max_{X_p \in \{X_a\}} \|\widehat{Y}(X_a) - \widehat{Y}(X_p)\|^2 - \min_{X_n \notin \{X_a\}} \|\widehat{Y}(X_a) - \widehat{Y}(X_n)\|^2\right)_+, \tag{59}$$

where the "+" subscript denotes the ReLU function. In practice, calculation of $\mathcal{L}_{trip}$ as defined in (59) is too costly. Various strategies may be used to circumvent this. According to [21], good performance is obtained when minibatch propagation is used, and the triplet loss function $\mathcal{L}_{trip}$ is evaluated a minibatch at a time. This leads to the following loss function, which in [21] is denoted $\mathcal{L}_{BH}$ (the "BH" stands for "batch hard"):

$$\mathcal{L}_{BH}(\mathcal{M}) = \sum_{X_a \in \mathcal{M}} \left(m + \max_{X_p \in \{X_a\} \cap \mathcal{M}} \|\widehat{Y}(X_a) - \widehat{Y}(X_p)\|^2 \right.$$
$$\left. - \min_{X_n \notin \{X_a\} \cap \mathcal{M}} \|\widehat{Y}(X_a) - \widehat{Y}(X_n)\|^2\right)_+. \tag{60}$$

The maxes and mins in (60) are much easier to compute than those in (59) because the only require working with input vectors in one minibatch at a time, rather than all the vectors in the training set.

If triplet loss is used, Euclidean distances are sufficient for classification and no softmaxing is necessary.

3.2.5 Loss Functions Based on Angular Distances

Recall the expression (48) for the cross-entropy loss when softmax is used (repeated here for convenience):

$$\mathcal{L}_{XE}(X, X^{(M-1)}) = -W^{(M)}_{\mathcal{I}(X)} \cdot X^{(M-1)} + \log\left(\sum_{j=1}^{J} e^{W_j^{(M)} \cdot X^{(M-1)}}\right). \tag{61}$$

In general, the inner product $W \cdot X$ between two vectors has a geometrical interpretation in terms of the vectors' lengths $\|W\|, \|V\|$ and the angle $\phi_{W,X}$ between them:

$$W \cdot X = \|W\| \|X\| \cos(\phi_{W,X}). \tag{62}$$

It follows from (61) that the cross-entropy loss will be largest if the cosine of the angle between the weight vector $W^{(M)}_{\mathcal{I}(X)}$ and the hidden-layer output vector $X^{(M-1)}$ is close to 1, i.e., the two vectors are pointed in nearly the same direction. From this point of view, it appears that vectors in the same output class will all tend to align along the same direction.

In order to build a margin into the classification, then it would appear that we should build an algorithm that enhances the tendency of same-class output vectors to align in the same direction. We may do this by modifying the loss function to rise more steeply with increasing angle (i.e., decreasing cosine). Different authors have chosen different strategies for doing this, as described below.

L-softmax loss is introduced in [32]. In accordance with the discussion above, the loss function is modified to be a steeper function of cosine, as follows:

$$\mathcal{L}_{LS}(X, X^{(M-1)}) = -\Psi_{LS} + \log\left(e^{\Psi_{LS}} + \sum_{j \neq \mathcal{I}(X)} e^{\|W_j^{(M)}\| \|X^{(M-1)}\| \cos(\phi_{W_j,X})}\right), \tag{63}$$

where

- $\Psi_{LS} := \|W^{(M)}_{\mathcal{I}(X)}\| \|X^{(M-1)}\| \psi(\phi_{W_{\mathcal{I}(X)},X})$;
- $\psi(\phi) := \cos(m(\phi - \text{floor}(\phi/\pi))) - 2m\,\text{floor}(\phi/\pi)$,
- m is an integer hyperparameter (usually $m = 2, 3$ or 4).

The function ψ is a continuous decreasing function with continuous derivative, which declines more steeply than the cosine function. The larger the value of m, the steeper the decline, and the larger the margin between classes.

A-softmax loss, introduced in [33], differs from L-softmax only in that the weight vectors $\{W_j^{(M)}\}$ are normalized to unit length prior to the computation of the loss function. The loss function for A-softmax is identical to (63), except all terms of the form $\|W\|$ are set equal to 1. A-softmax is used in the "SphereFace" facial recognition system: the name "SphereFace" makes reference to the fact that the weight vectors all have unit length, and thus lie on a hypersphere.

Large Margin Cosine Loss (LMCL) was introduced in [53]. The loss function is identical to the formula for L-softmax loss in (63), except all weight vectors are normalized to 1 (as in A-softmax) and Ψ_{LS} is replaced by Ψ_{LMCL} where

$$\Psi_{LMCL} = \|X^{(M-1)}\| \left(\cos(\phi_{W_{\mathcal{I}(X)},X}) - m\right); \tag{64}$$

and m is not necessarily an integer.

Additive Angular Margin Loss (AAML) was introduced in [11]. AAML is identical to LMCL except Ψ_{LMCL} is replaced by Ψ_{AAML} where:

$$\Psi_{AAML} = \|X^{(M-1)}\| \left(\cos(\phi_{W_{\mathcal{I}(X)},X} + m)\right), \tag{65}$$

where once again m is not necessarily an integer. This loss also improves computing time compared to A-softmax because $\cos(\theta + m)$ is easier to calculate from $\cos(\theta)$ than $\cos(m\theta)$.

3.3 Optimization of Learning Rate

Equation (20) includes the learning rate hyperparameter μ, which must be chosen by the user. Much research has gone into optimizing μ so as to improve performance of the algorithm. In this section we give two modifications that frequently lead to faster convergence.

3.3.1 Adaptive Gradient Descent (AdaGrad)

The MLP is being trained to identify features within input vectors. Some of these features may be much more frequent than others. This implies that some MLP parameters may undergo frequent adjustments during the training process, while others have much fewer opportunities for correction. Adaptive gradient descent (AdaGrad) is designed to equalize the learning rate for different parameters that experience different rates of adjustment [13]. AdaGrad's idea is to make larger updates for parameters that have undergone little change, and smaller updates for parameters that have undergone greater variations. Thus the learning rate is adapted to each of the parameters; it is greater for the least updated parameters, and lower for the most frequently updated parameters. This is accomplished by keeping a record of past parameter updates, as follows. Define

$$G_t^{(m)} := \sum_{s=1}^{t} g_s^{(m)} \odot g_s^{(m)}, \tag{66}$$

where $\odot$ is the element-wise multiplication. Note that $G_t^{(m)}$ increases as the index t increases, because all the terms in the summation in (66) are positive. In AdaGrad, (20) is then replaced by

$$\theta_{t+1}^{(m)} = \theta_t^{(m)} - \frac{\alpha}{\sqrt{G_t^{(m)} + \epsilon}} \odot g_t^{(m)}, \quad m = 1 \dots M, \tag{67}$$

(the ϵ is included to prevent division by 0). It follows that the effective learning rates (given by the coefficients of $g_t^{(m)}$) decrease over time (as the denominator grows larger). Furthermore the learning rates are different for each parameter: learning rates decrease more rapidly for parameters which have seen frequent and/or large adjustments during the training process. The parameters α and ϵ are chosen by the user.

AdaGrad has demonstrated better performance than stochastic gradient descent [18]. The main weakness of AdaGrad is related to its strength—since the effective learning rates always decreases, learning may slow down to the point where it effectively stops before an optimal solution is reached.

3.3.2 Delta Adaptive Gradient Descent (AdaDelta)

AdaDelta is an enhancement to AdaGrad that addresses the problem of decreasing learning rates. Rather than using a sum as in (66), AdaDelta relies on exponentially decaying averages. To explain this, we recursively define an exponential averaging operation $E_\gamma[\cdots]_t$ as follows:

$$E_\gamma[a]_1 := a_1; \quad E_\gamma[a]_t = (1-\gamma)E_\gamma[a]_{t-1} + \gamma a_t, \quad t = 2, 3, \dots. \tag{68}$$

Exponential averaging may be thought of as a weighted sum of terms, where recent terms have weights close to 1 and terms that are farther in the past have less and less weight. Because of the decay constant γ, the exponential average (unlike the summation in (66)) does not increase indefinitely. The AdaGrad update Eq. (67) is replaced by

$$\theta_{t+1}^{(m)} = \theta_t^{(m)} + \Delta\theta_t^{(m)}, \quad m = 1 \dots M, \tag{69}$$

where $\Delta\theta_t^{(m)}$ is defined recursively as:

$$\Delta\theta_t^{(m)} := -\sqrt{\frac{E_\gamma[\Delta\theta^{(m)} \odot \Delta\theta^{(m)}]_{t-1} + \epsilon}{E_\gamma[g^{(m)} \odot g^{(m)}]_t + \epsilon}} \odot g_t^{(m)}. \tag{70}$$

This recursive calculation means that only quantities from $t - 1$ need to be stored to compute the quantities for t: previous quantities do not need to be stored, so memory requirements are minimal. The inverse of this average of the previous gradients serves as a weighting of the learning rate, as it is the case with the adaptive gradient descent; while the numerator under the square root makes the units consistent. For a more thorough derivation, see [59].

Algorithm 1 Adaptive gradient descent Delta (AdaDelta)

```
Require: $\gamma$ = decay constant ; $\epsilon$ = small constant
1: Initialize $E_\gamma[\Delta\theta^{(m)} \odot \Delta\theta^{(m)}]_0 = 0$
2: for $t = 1$ to $T - 1$ do
3:     Compute $g_t^{(m)} \odot g_t^{(m)}$ and update $E_\gamma[g^{(m)} \odot g^{(m)}]_t$ using (68)
4:     Compute $\Delta\theta_t^{(m)}$ using (70)
5:     Compute $E_\gamma[\Delta\theta^{(m)} \odot \Delta\theta^{(m)}]_t$ using (68)
6:     Compute $\theta_{t+1}^{(m)}$ using (69)
7: end for
```

It should be noted that the sensitivity of AdaDelta to the hyperparameters (γ and ϵ) is relatively low compared to other methods. We can therefore set γ to 0.99 and ϵ to 10^{-6} as recommended in [59], and no hyperparameter optimization is required.

3.4 *Enhanced Training Techniques*

This section describes several techniques have been developed to improve the training of neural networks, leading to better classification accuracy.

3.4.1 Bagging

Inaccuracies in prediction or classification can arise when the training process causes the network to respond to features that are peculiar to the training data, which are not shared by data in general. Naturally, one way to counteract this is to find a larger and more representative dataset. But when more data is not available, the effect of peculiarities can be mitigated by selecting multiple training sets from the data at hand. Specifically, given a training set consisting of N vectors, several different training sets of the same size may be created by taking several random samples with the replacement of size N from the original training data (these are called *bootstrap samples*). These constructed datasets will have some duplicate vectors (because of the sampling with replacement), but they may provide enough variation to wean the network from peculiarities in the original dataset. Each bootstrap sample is used to independently train the network, yielding a different set

Algorithm 2 MLP with bagging pseudocode

```
1. Initialize: Set of $N$ training vectors; hyperparameter $B$=number of bootstrap samples (chosen by the user)
2. Training:
    (a) Draw $B$ bootstrap samples of size $N$ from the training set (with replacement)
    (b) Train the network independently with each bootstrap sample, producing $B$ different parameter sets $\{\Theta_1, \dots, \Theta_B\}$ that produce $B$ different outputs $\{Y_1(X), \dots, Y_B(X)\}$ for a given input vector $X$.
3. Output:
    (a) To predict data vector $X$, generate the $B$ outputs $Y_1(X), \dots Y_B(X)$ using the $B$ different parameters sets $\Theta_1, \dots, \Theta_B$
    (b) If the MLP outputs a classification, take the majority vote from $Y_1(X), \dots, Y_B(X)$; otherwise take an average of the outputs $Y_1, \dots, Y_B$ as the MLP output.
```

of network parameters (weights and biases) for each bootstrap sample. The training process and application are outlined in Algorithm 2:

This method of taking bootstrap samples and then averaging (or aggregating) the results is called "bootstrap aggregation," or *bagging* [3]. Bagging is a very general procedure and can be applied to any predictor, not just neural nets.

3.4.2 Boosting

Boosting is a way of giving a network-in-training a "second chance" on data vectors that it does not get right the first time. Like bagging, boosting involves multiple rounds of training, using sampling with replacement; but unlike bagging, the multiple samplings are done in sequence and not in parallel, because the probability weights for each sample depend on the training outcome of the previous sample. In the first training, all vectors in the training set have equal probability weight; but in subsequent training samples, vectors that were inaccurately predicted in the previous training round are given a heavier weight in the resampling.

A popular version of boosting is "AdaBoost" [14], which is described in the pseudocode in Algorithm 3.

3.4.3 Dropout

We have mentioned previously that overlearning is a serious and common pitfall which degrades neural network performance. Overlearning occurs when the weights on the architecture adapt to peculiar features within the training set that are not generalizable to other data. This can happen when groups of units develop overcomplicated responses to simple patterns. For example, suppose a net is being trained to identify even numbers, and the training set has no numbers that end

Algorithm 3 AdaBoost pseudocode

```
1. Initialize: Set $\mathcal{V}$ of $N$ training vectors, with equal probability weights $p_n = 1/N, n = 1 \dots N$;
   $M$ equals number of training iterations
2. For $m = 1$ to $M$:
   (a) Train the network, using training vectors that are sampled from $\mathcal{V}$ with replacement with
       selection probabilities $\{p_n\}$. The training result is a vector of MLP parameters $\Theta_m$, which
       produces output $Y_m(X)$ for a given input vector $X$.
   (b) Adjust the probability weights of training vectors $X$, according to how accurately they are
       predicted using $Y_m(X)$. If the prediction is accurate, the weight is reduced; otherwise the
       weight is increased.
   (c) Compute a weight $\alpha_m$ that reflects the overall accuracy of the predictor $Y_m$ when applied
       to the training set $\mathcal{V}$ sampled with weights $\{p_n\}, n = 1, \dots, N$.
   (d) Normalize the adjusted probability weights so they sum to one, and replace the previous
       set $\{p_n\}, n = 1, \dots N$ with the new set of weights
3. Output: For a given data vector input, the prediction depends on the weighted sum of the
   outputs:
```

$$weighted\_sum = \sum_{m=1}^{M} \alpha_m Y_m(X) \tag{71}$$

with the digit “8.” It is possible that the net may learn to individually identify even numbers with final digits 0, 2, 4, 6, without generalizing the rule to include the final digit 8 as well. To avoid situations like this, we should build some “fault tolerance” into the network, so that the network responds robustly. We may accomplish this by randomly deactivating different units during training, which is called *dropout* [46]. Training the net with dropout is almost identical to the process described in Sect. 2.2.3, except that units within the network are deactivated with probability $p < 1$, where p is a hyperparameter chosen by the user. If the original activation function for a unit is $f(V)$, under dropout the activation function becomes

$$f_{dropout}(V) = \begin{cases} 0 & \text{with probability } 1 - p; \\ f(V) & \text{with probability } p. \end{cases} \tag{72}$$

The result of dropout is that the layer activation vector functions $f^{(m)}$ in (11) are “pruned,” and some of the vector components in each layer are replaced with zeros. When backpropagation is performed on the MLP, the weights and biases for zeroed-out units are not modified: essentially, these units have been temporarily removed from the network. We emphasize that the pruning is different for each training vector, so that all of the units are still modified by the overall training process. At the end of the training period, all of the weights in the network are multiplied by p. This is because each unit has only participated in a proportion p of the training adjustments, so correspondingly the unit’s influence is reduced by a factor p.

It is also possible to perform dropout on the connections between units instead of (or in addition to) the units themselves. Dropout has been shown to bring large improvements to neural network performance [46].

L2 Regularization (Weight Decay)

Overlearning can occur when the MLP uses overly complex criteria to make predictions, rather than identifying simple common features. For example, to identify square shapes in a database of images of shapes the MLP could possibly use a complicated set of idiosyncratic features (size, contrast, color, shading, etc.) that are peculiar to the training dataset, rather than basing its decision on the actual shape itself. Just as in statistics, simpler is better: a model with fewer significant parameters is to be preferred. It is possible to nudge the MLP model in the direction of simplicity during the training process using a mathematical technique called *regularization*. A term is added to the loss function $\mathcal{L}$ that is proportional to the complexity of the weights in the MLP. Two common choices for this added term are:

L2 regularization: the loss function $\mathcal{L}$ is replaced by $\mathcal{L} + \frac{\lambda}{2}\sum_{m=1}^{M} \theta^{(m)} \cdot \theta^{(m)}$;

L1 regularization: the loss function $\mathcal{L}$ is replaced by $\mathcal{L} + \lambda \sum_{m=1}^{M} \theta^{(m)} \cdot \operatorname{sgn}(\theta^{(m)})$,

where

- $\theta^{(m)}$ is the vector of MLP parameters (introduced in Sect. 2.2.3);
- $\operatorname{sgn}(\cdot)$ is the *signum function*: $\operatorname{sgn}(x)$ is -1, 1, or 0 depending on whether x is negative, positive, or zero, respectively;
- λ is a hyperparameter chosen by the user, which controls the relative influence of the complexity term on the loss function;

These added terms have a very simple effect on the training: the only change is that Eq. (20) is slightly modified so that:

- *L2 regularization*: $\theta_{t+1}^{(m)} = (1-\lambda)\theta_t^{(m)} - \mu g_t^{(m)}$;
- *L1 regularization*: $\theta_{t+1}^{(m)} = \theta_t^{(m)} - \mu g_t^{(m)} - \lambda \operatorname{sgn}(\theta_t^{(m)})$.

In all other respects, the training proceeds exactly as described in Sect. 2.2.3. L2 regularization has the additional advantage that it penalizes larger weights more, so that the MLP weights become more equalized among connections.

4 Facial Recognition

Facial recognition is typically accomplished in three steps. First, the image is passed into a face detector that identifies and selects the areas of the image that contain faces. Next, the image areas containing faces are then reduced to descriptors in a more observable dimension. Finally, a classifier is responsible for predicting the individual according to the descriptors. Before 2014, the most powerful facial recognition systems were based on principal component analysis (PCA) and Bayesian models. Since 2014, however, most research has gravitated towards deep convolutional networks. This development was inspired by the extraordinary success of CNN architectures [16] in the ImageNet contest [9], such as AlexNet [26], VGGNet [45], GoogLeNet [51], and ResNet [20], among others. Because facial recognition is a special case of object recognition, good architectures for general object recognition can be used as base models for facial recognition. For instance, AlexNet is used in ([34, 42]), VGGNet in ([1, 7, 35, 39]), and GoogLeNet in ([43, 57]).

4.1 Convolutional Neural Net Models for Facial Recognition

In this section we briefly describe several successful CNN models used for facial recognition. Because of the rapid developments in the field, different models undergo continuous revision and consequently benchmark accuracies change considerably from year to year.

4.1.1 DeepFace

In March 2014, the Facebook AI research team announced the development of a facial recognition system that achieved near-human performance [52]. The model, named DeepFace, reached an accuracy of 97.25% on the LFW benchmark dataset (see Sect. 4.3), reducing error by more than 27% over the best previous models. The model was trained on a set of four million facial images belonging to over 4000 people.

One of the DeepFace's strengths is its front-face alignment technique based on 3D models. A 3D analytical model of the face based on landmarks is used to deform a detected face into a 3D front-end model [52]. This operation is carried out by an affine transformation T of X_{2d} to X_{3d}, driven by the Delaunay triangulation derived from 67 so-called fiducial points. The processing flow of the algorithm is shown in Fig. 11.

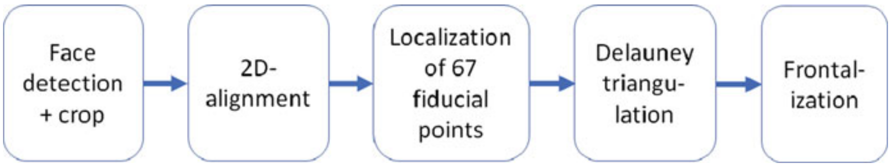


Fig. 11 Sequence of operations in DeepFace facial recognition system

4.1.2 DeepID (2015)

The work of DeepFace was extended by a series of articles describing DeepID [48], DeepID2 [47], DeepID2+ [50], DeepID3 [49], where each successive article improved the performance on the LFW data set. Unlike DeepFace, the DeepID systems do not use face alignment in 3D, but rather a simpler 2D affine alignment. Training of the DeepID series was done by combining the CelebFaces [31] and WDRef [5] datasets.

4.1.3 FaceNet (2015)

Google Inc. researchers have developed a deep CNN model called FaceNet [43] that makes use of the triplet loss function described in Sect. 3.2.4. The model held the record on the LFW benchmark until 2018, when it was replaced by ArcFace. FaceNet achieved 99.63% accuracy on the LFW dataset and 95.1% on YTF@@@ **YTF should be added to the facial recognition benchmarks section** @@@.

FaceNet utilized two different CNN architectures. The first built on the architecture of Zeiler and Fergus [60] that was originally used for image processing. This option has 22 layers with a total of 140 million parameters. The second option is based on the GoogLeNet architecture originally used for image processing [51]. The model consists of 17 layers and has a total of 7.5 million parameters, almost 20 times fewer than the first model.

4.1.4 VGGFace (2015)

The Visual Geometry Group (VGG) at Oxford University developed a facial recognition model simultaneously with the VGGFace database (see Sect. 4.3), which was used to show that VGGFace (a publicly available dataset) can train a network that can achieve comparable performance with models trained on proprietary datasets (such as FaceNet) [39]. Like FaceNet, the triplet loss function is used. Accuracies of 98.95% and 97.3% were achieved for lFW and YTF, respectively. Both the model and the database are implemented in Keras (keras.io), which is an open-source neural network library in Python.

4.1.5 SphereFace (2017)

SphereFace [33] uses the A-softmax loss function. In facial verification, SphereFace achieves an accuracy of 99.42% on LFW and 95% on YTF. Like VGGFace, SphereFace is trained on a public dataset (CASIA-WebFace—see Sect. 4.3).

4.1.6 CosFace (2018)

Like SphereFace, CosFace [53] was trained on CASIA-WebFace, but uses large margin cosine loss instead of A-softmax loss. Performance on LFW and YTF was 99.73% and 97.6%, respectively.

4.1.7 ArcFace (2018)

ArcFace [11], developed by researchers from Imperial College (London), achieved 99.83% accuracy on the LFW benchmark. ArcFace is based on an additive angular margin, and combines ideas from SphereFace and CosFace to generate normalized face cuts of size (112 $\times$ 112) using five facial landmarks. ResNet [20] is used as a template architecture.

4.2 Facial Recognition Without Constraint Using Deep Learning

Face recognition in real-world environments has significant intra-personal variations, such as exposure, lighting, expressions, and occlusions, which remains a big challenge for computer vision. This is the problem of facial recognition without constraint.

4.2.1 Data Variability Issues

Although deep learning approaches have been widely used because of their powerful representation, Ghazi et al. [37] have shown that various conditions such as poses, illuminations, expressions, and occlusions still significantly affect recognition performance. Two data transformation techniques that can be used to improve performance are *normalization* and *data augmentation*. Normalization refers to the process of taking the facial image to be recognized and using features to standardize the lighting and pose. Augmentation takes images in the dataset and makes modifications (for example, adding glasses, changing hairstyle, and/or pose) so that the identification is more robust to such variations.

4.3 Facial Recognition Datasets

There are a great many facial image datasets from multiple sources. In this section we limit ourselves to a few that are both publicly available and often referenced in facial recognition research for training or testing.

4.3.1 Labeled Faces in the Wild (LFW)

This dataset is a collection of JPEG images of faces of famous people collected on the internet [22, 27], and is designed to study the problem of facial recognition without constraint (i.e., no limitations are placed on lighting, expression, perspective, or other parameters). These faces were extracted from various online websites by face detectors based on the Viola–Jones model. LFW contains 13,233 images of 5749 tagged celebrities, where each image is centered on a single face. LFW is frequently used in competitions to measure algorithms' performance in facial recognition or facial verification.

4.3.2 CASIA-WebFace

CASIA-WebFace [58] (often referred to as WebFace) is a public dataset used for training purposes, consisting of nearly 500,000 images of about 10,000 subjects. Although it is smaller than MS Celeb-1M [17] or MegaFace1 [25] and much smaller than MegaFace2 [38], it is possible to train advanced models using this dataset [32, 55, 58].

4.3.3 VGGFace and VGGFace2

VGGFace [39] and its successor VGGFace2 [4] were developed by the Visual Geometry Group at the University of Oxford. The group's purpose was to develop a methodology for constructing a large-scale dataset which both includes diversity in pose and age for individual image subjects, and achieves a high degree of identification accuracy by making use of multiple stages of automatic and manual filtering. VGGFace (like CASIA-WebFace) is intended for training only, and includes 2.6 million images of 2622 subjects: a curated version (with identities manually validated) has 800,000 images. VGGFace2 includes 3.3 million images of 9131 subjects, and can be used for both training and testing.

4.3.4 Similar Looking Labeled Faces in the Wild (SLLFW)

The dataset SLLFW (formerly known as Fine-Grained Labeled Faces in the Wild, or FGLFW) [10] is a renovation of LFW. To enable the rigorous testing of algorithms' ability to verify faces, SLLFW selects face pairs from LFW from different individuals that closely resemble each other.

5 Conclusion

The application of deep learning has contributed remarkably to the current performance of facial recognition; and new elaborations and innovations are being introduced constantly, so that benchmark values rise every year. This chapter has some of the aspects that involved in this application, including CNN architecture, loss functions, training, and training datasets. We have also touched on the commonalities between facial recognition and the simpler problem of image type recognition.

References

1. W. AbdAlmageed, Y. Wu, S. Rawls, S. Harel, T. Hassner, I. Masi, J. Choi, J. Lekust, J. Kim, P. Natarajan et al., Face recognition using deep multi-pose representations, in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)* (IEEE, Piscataway, 2016), pp. 1–9
2. M.M. Adankon, Optimisation de ressources pour la sélection de modèle des SVM. PhD thesis, École de technologie supérieure, 2005
3. L. Breiman, Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
4. Q. Cao, L. Shen, W. Xie, O.M. Parkhi, A. Zisserman, Vggface2: a dataset for recognising faces across pose and age, in *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)* (IEEE, Piscataway, 2018), pp. 67–74
5. D. Chen, X. Cao, L. Wang, F. Wen, J. Sun, Bayesian face revisited: a joint formulation, in *European Conference on Computer Vision* (Springer, Berlin, 2012), pp. 566–579
6. C. Cortes, V. Vapnik, Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
7. N. Crosswhite, J. Byrne, C. Stauffer, O. Parkhi, Q. Cao, A. Zisserman, Template adaptation for face verification and identification, in *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)* (IEEE, Piscataway, 2017), pp. 1–8
8. G. Cybenko, Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**(4), 303–314 (1989)
9. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: a large-scale hierarchical image database, in *IEEE Conference on Computer Vision and Pattern Recognition, 2009. CVPR 2009* (IEEE, Piscataway, 2009), pp. 248–255
10. W. Deng, J. Hu, N. Zhang, B. Chen, J. Guo, Fine-grained face verification: FGLFW database, baselines, and human-DCMN partnership. Pattern Recogn. **66**, 63–73 (2017)
11. J. Deng, J. Guo, S. Zafeiriou, ArcFace: additive angular margin loss for deep face recognition (2018). Preprint. arXiv: 1801.07698
12. G. Dreyfus, *Apprentissage statistique* (Editions Eyrolles, Paris, 2008)

13. J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**(Jul), 2121–2159 (2011)
14. Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55**(1), 119–139 (1997)
15. K. Fukushima, S. Miyake, Neocognitron: a self-organizing neural network model for a mechanism of visual pattern recognition, in *Competition and Cooperation in Neural Nets* (Springer, Berlin, 1982), pp. 267–285
16. J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, L. Wang, G. Wang et al., Recent advances in convolutional neural networks (2015). Preprint. arXiv: 1512.07108
17. Y. Guo, L. Zhang, Y. Hu, X. He, J. Gao, MS-Celeb-1M: a dataset and benchmark for large-scale face recognition, in *European Conference on Computer Vision* (Springer, Cham, 2016), pp. 87–102
18. A.T. Hadgu, A. Nigam, E. Diaz-Aviles, Large-scale learning with AdaGrad on Spark, in *2015 IEEE International Conference on Big Data (Big Data)* (IEEE, Piscataway, 2015), pp. 2828–2830
19. R. Hadsell, S. Chopra, Y. LeCun, Dimensionality reduction by learning an invariant mapping, in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR'2006), vol. 2, pp. 1735–1742
20. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778
21. A. Hermans, L. Beyer, B. Leibe, In defense of the triplet loss for person re-identification (2017). Preprint. arXiv: 1703.07737
22. G.B. Huang, M. Mattar, T. Berg, E. Learned-Miller, Labeled faces in the wild: a database for studying face recognition in unconstrained environments, in *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition* (2008)
23. D.H. Hubel, T.N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J. Physiol. **160**(1), 106–154 (1962)
24. K.Th. Kalveram, A modified model of the Hebbian synapse and its role in motor learning. Hum. Mov. Sci. **18**(2–3), 185–199 (1999)
25. I. Kemelmacher-Shlizerman, S.M. Seitz, D. Miller, E. Brossard, The MegaFace benchmark: 1 million faces for recognition at scale, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4873–4882
26. A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
27. E. Learned-Miller, G.B. Huang, A. RoyChowdhury, H. Li, G. Hua, Labeled faces in the wild: a survey, in *Advances in Face Detection and Facial Image Analysis* (Springer, Cham, 2016), pp. 189–248
28. G. Lebrun, Sélection de modèles pour la classification supervisée avec des SVM (Séparateurs à Vaste Marge). Application en traitement et analyse d'images. PhD thesis, Université de Caen Basse-Normandie, 2006
29. Y. LeCun, B.E. Boser, J.S. Denker, D. Henderson, R.E. Howard, W.E. Hubbard, L.D. Jackel, Handwritten digit recognition with a back-propagation network, in *Advances in Neural Information Processing Systems* (1990), pp. 396–404
30. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. Nature **521**(7553), 436 (2015)
31. Z. Liu, P. Luo, X. Wang, X. Tang, Deep learning face attributes in the wild, in *Proceedings of International Conference on Computer Vision (ICCV), December* (2015)
32. W. Liu, Y. Wen, Z. Yu, M. Yang, Large-margin softmax loss for convolutional neural networks, in *ICML* (2016), pp. 507–516
33. W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, L. Song, SphereFace: deep hypersphere embedding for face recognition, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1 (2017), p. 1
34. X. Liu, M. Kan, W. Wu, S. Shan, X. Chen, VIPLFaceNet: an open source deep face recognition SDK. Front. Comput. Sci. **11**(2), 208–218 (2017)

35. I. Masi, A.T. Tran, T. Hassner, J.T. Leksut, G. Medioni, Do we really need to collect millions of faces for effective face recognition? in *European Conference on Computer Vision* (Springer, Cham, 2016), pp. 579–596
36. W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**(4), 115–133 (1943)
37. M. Mehdipour Ghazi, H.K. Ekenel, A comprehensive analysis of deep learning based representation for face recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2016), pp. 34–41
38. A. Nech, I. Kemelmacher-Shlizerman, Level playing field for million scale face recognition, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, Piscataway, 2017), pp. 3406–3415
39. O.M. Parkhi, A. Vedaldi, A. Zisserman, Deep face recognition, in *British Machine Vision Conference*, vol. 1 (2015), p. 6
40. C. Qi, F. Su, Contrastive-center loss for deep neural networks, in *2017 IEEE International Conference on Image Processing (ICIP)* (IEEE, Piscataway, 2017), pp. 2851–2855
41. F. Rosenblatt, The perceptron: a perceiving and recognizing automaton. Technical report, Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957
42. S. Sankaranarayanan, A. Alavi, C. Castillo, R. Chellappa, Triplet probabilistic embedding for face verification and clustering (2016). Preprint. arXiv: 1604.05417
43. F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: a unified embedding for face recognition and clustering, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 815–823
44. A. Sidani, T. Sidani, A comprehensive study of the backpropagation algorithm and modifications, in *Conference Record Southcon* (IEEE, Piscataway, 1994), pp. 80–84
45. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition (2014). Preprint. arXiv: 1409.1556
46. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
47. Y. Sun, Y. Chen, X. Wang, X. Tang, Deep learning face representation by joint identification-verification, in *Advances in Neural Information Processing Systems* (2014), pp. 1988–1996
48. Y. Sun, X. Wang, X. Tang, Deep learning face representation from predicting 10,000 classes, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 1891–1898
49. Y. Sun, D. Liang, X. Wang, X. Tang, Deepid3: face recognition with very deep neural networks (2015). Preprint. arXiv: 1502.00873
50. Y. Sun, X. Wang, X. Tang, Deeply learned face representations are sparse, selective, and robust, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 2892–2900
51. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1–9
52. Y. Taigman, M. Yang, M. Ranzato, L. Wolf, DeepFace: closing the gap to human-level performance in face verification, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 1701–1708
53. H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, W. Liu, CosFace: large margin cosine loss for deep face recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 5265–5274
54. K.Q. Weinberger, J. Blitzer, L.K. Saul, Distance metric learning for large margin nearest neighbor classification, in *Advances in Neural Information Processing Systems* (2006), pp. 1473–1480
55. Y. Wen, K. Zhang, Z. Li, Y. Qiao, A discriminative feature learning approach for deep face recognition, in *European Conference on Computer Vision* (Springer, Cham, 2016), pp. 499–515

56. P.J. Werbos, Backpropagation through time: what it does and how to do it. Proc. IEEE **78**(10), 1550–1560 (1990)
57. J. Yang, P. Ren, D. Zhang, D. Chen, F. Wen, H. Li, G. Hua, Neural aggregation network for video face recognition, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, Piscataway, 2017), pp. 5216–5225
58. D. Yi, Z. Lei, S. Liao, S.Z. Li, Learning face representation from scratch (2014). Preprint. arXiv: 1411.7923
59. M.D. Zeiler, ADADELTA: an adaptive learning rate method (2012). Preprint. arXiv: 1212.5701
60. M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in *European Conference on Computer Vision* (Springer, Cham, 2014), pp. 818–833

Improving Deep Unconstrained Facial Recognition by Data Augmentation

Arnauld Fountsop Nzegha, Jean Louis Ebongue Fendji, Christopher Thron, and Clementin Djameni Tayou

1 Introduction

Facial recognition technology aims to recognize the face of a (human or animal) subject by using biometrics to map facial features from a picture. This recognition is usually achieved through artificial intelligence techniques, mainly deep learning.

Deep learning is a subfield of machine learning which is based on hierarchical learning architectures for data representation [25]. It includes a set of methods that represent data with a high level of abstraction through nonlinear transformations. One of the strengths of deep learning lies in its ability to exploit technological advances in computing power. Today, deep architectures based on convolutional neural networks (CNN) form the basis of most facial biometrics technology, and are very robust [27, 29, 34, 37, 38, 38–41].

Many CNN models can achieve close to 100% accuracy on standard benchmarks such as Labeled Faces in the Wild (LFW), MegaFace, or YouTubeFace (YTF). However, these models cannot guarantee the same performance in a realistic environment which is uncontrolled. Facial recognition remains a big challenge when it is in a non-constrained scenario or without the cooperation of the subject. Indeed, the face of a person can be very different observed on images taken in various poses, lighting conditions, at different ages, with makeup or different

A. F. Nzegha (✉) · C. D. Tayou
Department of Mathematics and Computer Science, University of Dschang, Dschang, Cameroon

J. L. E. Fendji
Department of Computer Engineering, Institute of Technology, University of Ngaoundere, Ngaoundere, Cameroon

C. Thron
Department of Science and Mathematics, Texas A&M University-Central Texas, Killeen, TX, USA

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_7

facial expressions. These hazards lead to a great intra-personal variability which is the major difficulty of facial recognition without constraint. In particular, lighting variations can significantly reduce the performance of face recognition systems. The present chapter tackles this issue by using data augmentation that makes it possible to overcome the problem of intra-personal variability by inserting more diversity in the facial recognition data. Lighting compensation allows facial recognition models to better capture the dynamics of lighting and greatly improves recognition.

This chapter is organized as follows. Section 2 describes elements of the system design, including the image processing techniques used for introducing lighting variation as well as techniques used to train the CNN. Section 3 describes the evaluation of the approach using a convolutional neural network model inspired by VGGNet-16 [35], that was trained on the LFW dataset and then tested on the YaleB and ORL databases. Section 4 presents and discusses the results of the experiments described in Sect. 3; and Sect. 5 gives some concluding remarks.

2 Facial Recognition System Design Elements

2.1 Overview

The implementation of our improved facial recognition system can be divided into two main tasks: data augmentation and CNN training. These two main tasks may be subdivided as follows.

Data augmentation is accomplished by modifying images in the training set so as to simulate the effects of different lighting situations on the appearance of the face in the image. This simulation requires first that a 3D reconstruction of the face be inferred from the image and then a lighting model applied to the 3D reconstruction to perform a pixel-by-pixel alteration of the apparent brightness of the image pixels.

CNN training involves using training sets to train the CNN to make a correct identification when presented with a facial image of one of a limited number of subjects that are known to the CNN. This training also proceeds in two stages. First, a large training set from a wide variety of faces is used to train the CNN to recognize distinguishing "features," which help to uniquely identify the image. This dataset is augmented by the data augmentation procedures described above, to improve the CNN's robustness to variations in lighting. After this, a smaller dataset containing only images of the specific subjects to be identified (in various poses and lighting conditions) is used to familiarize the CNN so that it can make specific identifications.

The techniques used in the steps outlined above are described in the following subsections.

2.2 *Data Augmentation*

2.2.1 Data Augmentation Overview

Data augmentation attacks the problem of lighting variation from a different angle. Rather than modifying the original image before attempting recognition, data augmentation uses the unmodified image but increases the variation in the training set. This makes the system training more rigorous, and the trained network is better able to cope with lighting variations.

Early versions of augmentation for lighting variations used 2-D methods. The 2012 AlexNet and 2014 VGGNet systems for image recognition used principal component analysis (PCA) on the set of RGB values for all pixels and all images in the training set. Based on the eigenvalues and eigenvectors from PCA, a (mean-zero multivariate Gaussian distribution) was constructed in RGB space, and additional images were generated from a given training image by selecting a single vector from this distribution and adding it to the RGB values for all pixels of the given image [22, 35].

More recently, data augmentation methods for lighting compensation have been based on 3-D reconstructions, which enable more accurate estimation of the effects of lighting variation. Examples include Paysan et al. [30] and Jiang et al. [19]. Sixt et al. [36] propose a new framework called RenderGAN that can generate large datasets of realistic labeled images by combining a 3-D model and a GAN model.

2.2.2 3-D Face Reconstruction

3-D face reconstruction from a 2-D image is an important, long-standing problem in facial recognition. Apart from lighting compensation, 3-D facial surface models have been used in other applications such as 3-D facial expression recognition [42] and facial animation [31].

Point Clouds

Point clouds are the simplest type of surface representation, and forms the basis for most 3-D object representation methods. A point cloud consists of an unordered set of 3-dimensional vectors that represent points lying on the surface. As described below, more descriptive surface representations (e.g., mesh representations) are typically constructed based on point cloud representations.

Polygonal Meshes

In 3-D research in general and 3-D facial recognition in particular, most researchers represent the surfaces of 3-D objects as meshes. A mesh is essentially an unordered set of vertices (points), edges (connections between two vertices), and faces (sets of edges with shared vertices) that together represent the surface explicitly. Generally, the faces consist of triangles, quadrilaterals, or other simple convex polygons, which simplifies the rendering. The task of building a mesh from a cloud of points is commonly called the surface reconstruction problem. Several techniques exist, many of which are based on the classic method of Delaunay triangulation [26].

Polygonal mesh surface representations provide several distinct advantages over point clouds. They enable much clearer representations of the surface using 3-D plotting software. Numerous ray tracing, collision detection, and rigid body dynamics algorithms have been developed for polygonal meshes. They provide explicit knowledge of connectivity, which can be used to calculate the geodetic distance between points on the surface. This is particularly useful in face recognition, because geodetic distances are more invariant than Euclidean distances under changes of expression by the subject [7]. Some researchers have exploited this by directly comparing point to point distances, or by using isogeodesic curves [18].

Meshes do have a significant drawback in that they may have errors such as cracks, holes, T-junctions, overlapping polygons, duplicate geometry, and auto-intersections. These defects may impede the mesh path and affect the quality of rendering. To correct these problems, either a local or global approach may be used, depending on the seriousness of the defects[2]. The local approach locates each specific defect in the mesh and tries to fix it while conserving the model unchanged, and consequently preserves most of the mesh information unchanged. It is suitable when meshes have rare defects and consist of operations such as triangulation for cracks, or filling holes [15], etc. On the other hand, the global approach takes into account both individual defects and the mutual relations between defects, and requires some adjustment for most or all mesh elements.

In cases where lighting of the surface is being considered, the mesh must include orientation information of the mesh faces. The orientation of a flat face is given by a vector of unit length that is normal (i.e., perpendicular) to the face. Any face has two unit normal vectors that point in opposite directions. In cases when the mesh is describing a human face, one normal points "outward" and the other points "inward." As we shall see in Sect. 2.2.3, the brightness of a surface depends on the angle between the surface's outward normal and the lighting source. It follows that in order to correctly predict brightness, the outward normal must be correctly identified. Algorithms have been designed to consistently identify outward normals: for example, Borodin et al. [6] propose and verify an algorithm for consistently orienting the normals of a boundary representation, even in the presence of gaps, T-junctions, and intersections.

3D Morphable Models

3D morphable models (3DMM) were introduced in 1999 by Blanz and Vetter [5]. They first created a database of 200 3-D face scans in standardized position using Cyberware^{TM} laser scan technology, which captures comprehensive 3-dimensional geometric and textural data. The i'th face in the database is represented as a pair of $3n$-dimensional vectors $(\mathbf{S}_i, \mathbf{T}_i)$, where $\mathbf{S}_i$ and $\mathbf{T}_i$ capture the geometric and texture (RGB) information, respectively. Additional faces can then be modeled as "morphs" of the faces in this database, which represented as linear combinations of the face vectors in the database. The mathematical expression is

$$(\mathbf{S}_{morph}, \mathbf{T}_{morph}) = (\sum_{i=1}^{m} a_i \mathbf{S}_i, \sum_{i=1}^{m} b_i \mathbf{T}_i), \tag{1}$$

where m is the number of faces in the database, and $\{a_i\}$ and $\{b_i\}$ are sets of coefficients such that $\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} b_i = 1$ (coefficients can be positive or negative). The two sets of coefficients reflect the fact that geometry and texture are modeled independently.

UV-Mapping and 3D Face

3D models must contain both 3D shape and texture information. Typically, the shape information is given by a 3D mesh as described above. Since the texture information describes properties of a surface, the texture information is inherently two-dimensional and may be stored in a two-dimensional array. However, since the surface is not flat, a mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ is required to associate two-dimensional points with the actual positions in 3-dimensional space. This mapping parametrizes the facial surface in terms of a pair of coordinates, which are generally denoted as (U, V). The texture values (R, G, B) are then stored as functions of (U, V). Feng et al. [12] have developed a facial recognition system that uses a CNN to regress the (U, V) position map directly from unconstrained 2D images. Their approach attains state-of-the-art performance, with much lower processing times than other methods. In their system, the reconstructed 3D face is represented by:

- 43868 3D vertices, where each vertex has x, y, z coordinates.
- 43868 UV mapping coordinates corresponding to the 3D vertices.
- 86907 triangular faces, which form a triangulation of the 3D vertices.
- UV texture information of size $256 \times 256 \times 3$ (RGB values for a 256×256 grid in the UV plane).

2.2.3 Lighting Variation

In this section we describe the modification of images used to augment the dataset in order to account for lighting variations. We assume that our face surface is a Lambertian surface, so can reflect the light following the Lambert reflectance model [4]. The overall process for introducing lighting variation is shown in Fig. 1.

The first step in the process shown in the figure is to calculate the normal vectors at all vertices of the 3D triangulation, which may be used to calculate the light

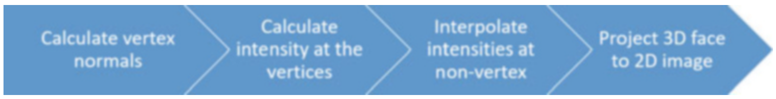

Calculate vertex normals
Calculate intensity at the vertices
Interpolate intensities at non-vertex
Project 3D face to 2D image

Fig. 1 Process of illumination variation

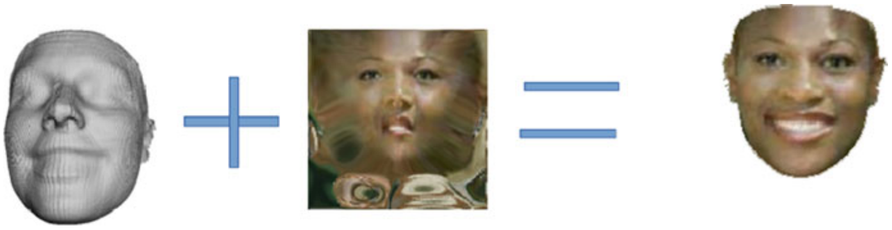


Fig. 2 Fusion of the mesh and the UV-texture

intensity at each vertex. Lighting values at surface points on the triangular faces can then be interpolated based on vertex values using barycentric coordinates. In order to be able to calculate the vertex normals, we first calculate unit normal vectors (i.e., perpendiculars) to the triangular surfaces that meet at the vertex. For a triangular surface F of vertices (v_1, v_2, v_3), a normal vector to the plane of F may be calculated as a cross product of two edge vectors, $N_f = \overrightarrow{v_1 v_2} \times \overrightarrow{v_1 v_3}$ (the vertices in the mesh must be correctly ordered so that the outward rather than the inward normal is produced, as discussed in Sect. 2.2.2).

Given N_f, the unit outward normal vector to a surface is found as

$$\hat{N_f} = \frac{N_f}{|N_f|}. \tag{2}$$

We estimate the unit normal at the vertex v_i by the normalized sum of the normals to the surfaces containing v_i as follows:

$$N_{v_i} = N(v_i) = \frac{\sum \widehat{N}_f}{|\sum \widehat{N}_f|} \quad \text{for all} f \text{such that } v_i \in f. \tag{3}$$

To calculate the intensities at the vertices, we merge the 3D shape and the UV-texture by using the UV-mapping. This consists in associating the vertex v_i to the corresponding pixel (U_i, V_i) in the UV plane (Fig. 2). The intensity information $I(U_i, V_i)$ is used as the initial intensity for vertex v_i.

Compute the Intensities

In 3D space, we randomly choose a point $L_{x,y,z}$ as a lighting source. The unit direction vector from L to the vertex v_i is denoted by $l_i = \overrightarrow{Lv_i}/|\overrightarrow{Lv_i}|$. We then apply the Lambert reflectance model, so that the modified intensity of v_i is given by

$$I(v_i) = I(U_i, V_i) \cdot I_{l_i} cos(\theta), \tag{4}$$

where I_{l_i} is the relative intensity of the lighting source and θ is the angle between l_i and N_{v_i}. In our model, I_{l_i} was chosen as the constant value 1.5 independent of i.

The cosine function is a decreasing function. If the angle between θ is too large, the intensity will vanish or even become negative. To avoid the loss of information, we add to the new intensity 50% of the initial intensity.

$$I(v_i) = I_0(v_i) I_{l_i} cos(\theta) + \frac{1}{2} I_0(v_i) \tag{5}$$

This equation was used to compute the intensity for all vertices. For each triangle in the mesh, 13 additional points $P_1, \dots P_{13}$ are created within the triangle according to the equation:

$$P_i = \sum_{j=1}^{3} w_{ij} v_j = 1, i = 1 \dots 13, \tag{6}$$

where v_1, v_2, v_3 are the vertices of the triangle and $\{w_{i1}, w_{i2}, w_{i3}\}_{i=1,\dots 13}$ represent 13 sets of barycentric weights where $w_{ij} \geq 0$ and $\sum_{j=1}^{3} w_{ij} = 1, i = 1 \dots 13$. The intensity at point P_i is then calculated as

$$I(P_i) = \sum_{j=1}^{3} w_{ij} I(v_j). \tag{7}$$

The 3D face has three coordinates (x, y, z), where z represents the depth. Given that the 3D face is aligned, the coordinates (x, y) give a 2D plane containing the face (Fig. 3).

2.3 CNN Training for Classification

2.3.1 Overview

Face recognition (and object classification in general) can be separated into two steps: features extraction and classification. The two tasks can be performed separately: using transfer learning, a pre-trained feature extractor can be used, associated with an MLP or another classifier for classification, as described in the following subsections.

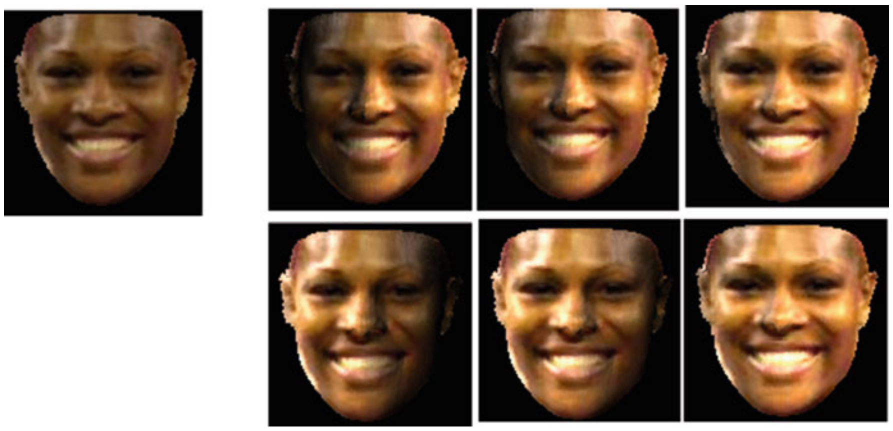


Fig. 3 Illustration of illumination variation

2.3.2 Features Extraction

Features extraction consists of finding a set of quantities calculated from a dataset (called features) that capture the datasets' essential characteristics. Classical methods of feature extraction include PCA [20] and linear discriminant analysis (LDA) [3]. CNN has recently emerged as the most successful feature extractor for images [44], etc. With CNN, an image is passed through a succession of filters arranged in layers which successively transform and reduce the data, creating representations of the image called *feature maps*. These feature maps can be concatenated or "flattened" into a single vector, called a *features vector*.

Classification

The classification step assigns the image to a predefined class, based on the features vector that is calculated by the features extractor. Classifiers in the literature include k-nearest neighbors (KNN) [43], support vector machines (SVM) [10], decision trees [32], and multilayer perceptron (MLP) [11]. In this work we use the MLP, which is a classifier based on neural networks, where the neurons of one layer are fully connected to those of the next layer.

Transfer Learning

A challenge of image classification (and classification tasks in general) is to train a classifier for images from a particular source domain when representative training data is scarce. Sometimes it is possible to find a related domain where sufficient training data is available to develop an effective classifier. In such cases, it is reasonable to suppose that a feature extractor that works well in a related domain will also work well in the source domain of interest. All that remains is to replace the classifier for the related domain with another that is particularly adapted to the source domain. This technique of reusing feature extractors is called *transfer learning* [28], and is a widely used strategy in the field of machine learning.

3 Experimental Setup

3.1 Computational Platform

For our system we used the Google Colaboratory platform [8], which is a cloud service based on Jupyter Notebooks designed support machine learning. It provides a fully configured runtime environment for deep learning, as well as free access to a robust graphics processor and TPU (tensor processing units). Currently it has NVIDIA GPUs with a preinstalled CUDA environment and several machine learning frameworks including TensorFlow, Theano, and scikit-learn. We used the TensorFlow framework [1] to train our model. TensorFlow is a programming framework for numerical computation that was made open source by Google Brain in November 2015. By 2017, TensorFlow was the most popular open source machine learning project on GitHub, even though it had only been available for little over a year [23]. To date it is one of the most powerful tools for Deep Learning, largely because of the ease of manipulation of tensors and their parallelization by tools such as CUDA.

Dlib-ml [21] is a state-of-the-art C++ toolkit containing machine-learning algorithms and data analysis tools, intended for both engineers in industry and researchers. We used Dlib to detect and extract the area of the face. This detector offers two methods for face detection: histogram of oriented gradients and support vector machine (HOG + SVM), and CNN-based detection. We used CNN-based detection: despite the heavy computing power required, since it is more suitable for non-frontal face detection. Results from [9] showed that the cascade algorithms with a CNN was superior to HOG + SVM or Haar Cascade methods in terms of both accuracy and speed criteria in unconstrained face detection problems.

PRNet (position map regression network) is an implementation of the system of Feng et al. [12] for regression of 3DMM model parameters. Initially this TensorFlow-based library was designed for pose estimation, facial alignment, and texture editing on the basis of 3D representation of the face. We use PRNet for 3D reconstruction of the different faces of the LFW dataset.

3.2 Description of CNN Model

We use a CNN for features extraction, therefore a stack of convolutional layers alternated with the pooling layers.

3.2.1 Inputs

We worked with grayscale images of size 100×100. All outputs from all convolutional layers undergo linear rectification using the ReLU activation function.

3.2.2 Filters

We use 2D convolutional filters of size 3×3, since the inputs are not very large. Small filters make it possible to detect highly localized patterns. The first convolutional layer has 64 filters applied to the input image, and the number of filters is doubled after each pooling layer. This process was inspired by the VGG16 [35] network of the group *Visual Geometric Group* which has a similar architecture. Unlike VGG16, we added a batch normalization [17] before every convolutional layer to avoid internal covariate shift which can slow down training and degrade performance. We also used dropout for regulation and greater robustness. Since [14] specifies that dropout has limited benefits when applied to convolutional layers, we applied dropout only on the first fully connected layer.

3.2.3 Subsampling (Pooling)

The feature extractor is subdivided into five blocks of convolutional layers separated by pooling layers. The first two blocks have two convolution layers followed by a pooling layer, because these layers produce larger feature maps and require more pooling. After this, subsequent blocks stack four convolutional layers before pooling. We chose to use 2×2 max pooling, which decreases each dimension of the feature maps by two (Fig. 4).

For identification, we use two fully connected layers of 1024 units each. A final softmax layer produces outputs in the interval [0, 1]: the number of neurons in the final layer is equal to the number of labels in the dataset. The outputs are interpreted as posterior probabilities of each individual. After the first fully connected layer we insert a dropout function that randomly eliminates 25% of the layer's outputs.

System hyperparameters are summarized in Table 1.

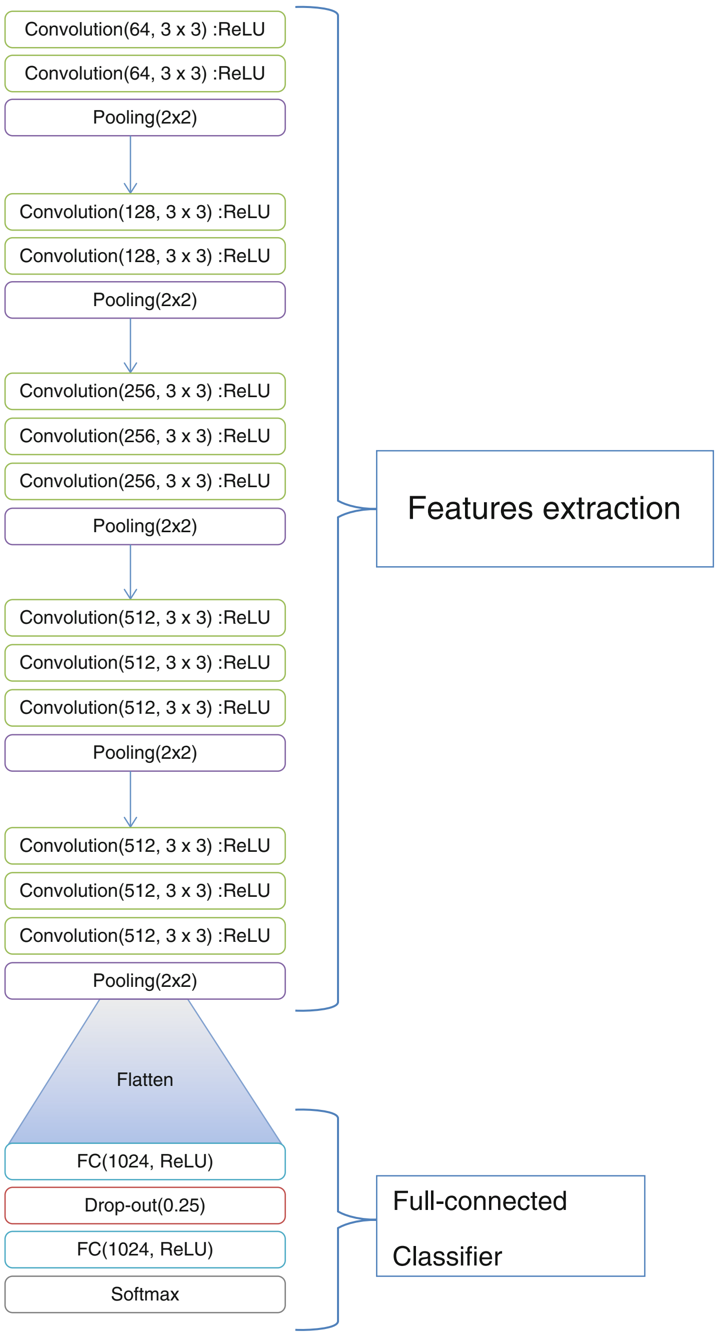

Convolution(64, 3 x 3) :ReLU
Convolution(64, 3 x 3) :ReLU
Pooling(2x2)
Convolution(128, 3 x 3) :ReLU
Convolution(128, 3 x 3) :ReLU
Pooling(2x2)
Convolution(256, 3 x 3) :ReLU
Convolution(256, 3 x 3) :ReLU
Convolution(256, 3 x 3) :ReLU
Pooling(2x2)
Convolution(512, 3 x 3) :ReLU
Convolution(512, 3 x 3) :ReLU
Convolution(512, 3 x 3) :ReLU
Pooling(2x2)
Convolution(512, 3 x 3) :ReLU
Convolution(512, 3 x 3) :ReLU
Convolution(512, 3 x 3) :ReLU
Pooling(2x2)
Features extraction
Flatten
FC(1024, ReLU)
Drop-out(0.25)
FC(1024, ReLU)
Softmax
Full-connected
Classifier

Fig. 4 Proposed model of convolutional network. The number of softmax outputs is set equal to the number of distinct individuals to be identified

Table 1 Setting of CNN

Hyperparameters	Details
Activation	ReLU
Initialization of weights	Random uniform
Optimizer	Adam
Number of iteration on LWF	1000
Number of iteration on YaleB and ORL	500

3.3 Datasets Used

3.3.1 Labeled Faces in the Wild (LFW)

This dataset is a collection of JPEG images of the faces of famous people collected on the internet [16, 24]. These faces were extracted from various online websites by face detectors based on the Viola–Jones model. LFW is commonly used to evaluate the performance of facial recognition systems. It contains 13,233 250×250 images of 5749 tagged celebrities, where each image is centered on a single face. In our experiments, the LFW dataset was used for pretraining the model. The data augmentation described in Sect. 2.2 was applied to these images to improve robustness of the feature extractor to lighting variations. The feature extractor was retained, and the classifier was replaced with classifiers for the smaller datasets described below. Since the smaller datasets have grayscale images, the LFW images were converted to grayscale using the function *cv2.cvtColor* from openCV.

3.3.2 ORL Database

The ORL Database [33] from the University of Cambridge contains 400 112×92 pixel images of 40 people, with 10 images per person. The images were taken at different times, with differing lighting and facial expressions. All the images were taken on a dark and homogeneous background, the subjects being in a frontal position, with a tolerance for certain lateral movements. In our experiment, the ORL database, like the YaleB database, was used for retraining the classifier.

3.3.3 Yale Face Database B

YaleB [13] was developed to allow systematic testing of facial recognition methods under wide variations in lighting and pose. It contains 5760 640×480 pixel images of 10 subjects viewed each in 576 viewing conditions (9 poses $\times$ 64 lighting conditions). For each subject in a particular pose, an image with ambient lighting (background) was also captured. In our experiment, a selection of images from YaleB was used for retraining the classifier, according to the transfer learning methodology described in Sect. 2.3.

3.4 Experimental Training and Testing Configurations

3.4.1 Experiment 1: LFW Without Data Augmentation

For the first experiment, we randomly selected 12,600 images from the LFW database for training, corresponding to 5749 subjects. For the initial feature extractor and classifier training, a classifier with 5749 softmax outputs was trained over 1000 iterations. Of the training set images, 10% (1260 images) were used for cross-validation.

After this, the classifier was removed and replaced by a classifier for ORL data and then for YaleB data. The new classifiers had the same number of layers as the original, but fewer softmax outputs (40 outputs for ORL, 10 outputs for YaleB). For training the ORL classifier, from the 400 images in the database 300 images were used for training and 100 for validation over 500 iterations. For training the YaleB classifier, 8 images per subject were used for training and 3 images per subject were used for validation, over 500 iterations. The three images chosen for evaluation had very uneven light distributions: one illuminated only on the left, one only on the right, and a third image with a central lighting.

3.4.2 Experiment 2: LFW with Data Augmentation

The second experiment differed from the first only in the training set used for the initial feature extractor + classifier training. The training set used 700 images from LFW of 28 subjects (25 images per subject). Each of these images was subjected to 18 different lighting conditions (representing a range of lighting directions, from left to right), for a total of 12,600 images. Since 28 subjects were used, the classifier in this case had 28 outputs. As in Experiment 1, after pretraining the classifier was replaced with a classifier of 40 outputs for ORL and with a classifier of 10 outputs for YaleB.

4 Results and Interpretation

4.1 Evaluation on ORL

Figures 5 and 6 show the training and testing accuracy curves for ORL data in Experiment 1 (without augmentation) and Experiment 2 (with augmentation), respectively. The figures show roughly 10% improvement in accuracy when augmented data is used for pretraining.

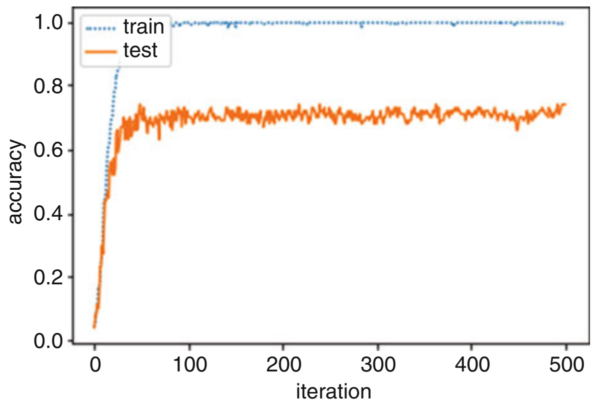


Fig. 5 Accuracy curve using ORL data for the model pretrained with plain LFW data (Experiment 1)

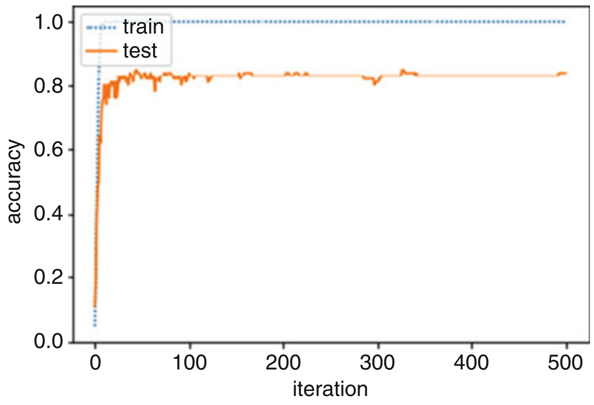


Fig. 6 Accuracy curve using ORL data for the model pretrained with augmented LFW data (Experiment 2)

4.2 Evaluation on YaleB

Figures 7 and 8 show the training and testing accuracy curves for YaleB data in Experiment 1 (without augmentation) and Experiment 2 (with augmentation), respectively. In this case, the improvement in test accuracy when augmented data is used for pretraining is about 20%.

Table 2 summarizes the results from the two experiments. For the evaluation of the model on test data, we observe a gain of accuracy of 9% on ORL. This finding confirms the enhanced effectiveness of pretraining that uses more images of fewer subjects taken under variable lighting conditions. The 18% improvement on YaleB shows that the performance improvement is further amplified in cases where the system is applied to images that also have variable lighting. In both experiments, the accuracy on YaleB is lower than the corresponding accuracy on ORL: this is probably due to the greater variability of both position and lighting for images in YaleB, as well as the smaller size of the dataset used in our experiments.

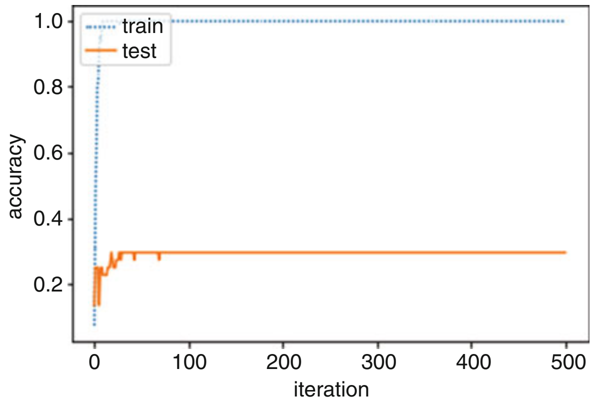


Fig. 7 Accuracy curve using YaleB for the model pretrained with plain LFW data (Experiment 1)

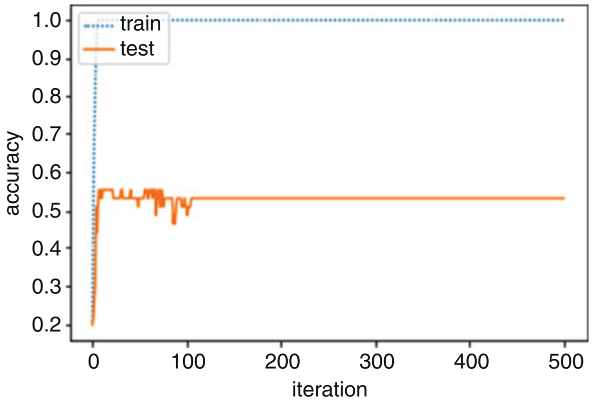


Fig. 8 Accuracy curve using YaleB for the model pretrained with augmented LFW data (Experiment 2)

Table 2 Summary of accuracies and average errors observed for the ORL and YaleB data sets, with and without data augmentation

	ORL		YaleB	
	Accuracy	Average error	Accuracy	Average error
Simple data	74%	1.78	35.56%	6.03
Augmented data	83%	0.72	53.33%	3.12

5 Conclusion

This chapter demonstrates a practical method for improving deep facial recognition using data augmentation. Specifically, we employed 3D lighting variation as a method of data augmentation, using the Lambert reflectance model to model the dynamics of ambient lighting in 3D space. The improvement was verified on two different datasets possessing different degrees of image variability. Accuracy gain due to data augmentation ranged from 9% to 18%, with the greater gain observed in the dataset that showed more variability. We conclude that lighting variation using the Lambert reflectance model is well suited to data augmentation for deep facial recognition under unconstrained lighting conditions.

References

1. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., TensorFlow: a system for large-scale machine learning, in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)* (2016), pp. 265–283
2. M. Attene, M. Campen, L. Kobbelt, Polygon mesh repairing: an application perspective. ACM Comput. Surv. (CSUR) **45**(2), 15 (2013)
3. S. Balakrishnama, A. Ganapathiraju, Linear discriminant analysis-a brief tutorial. Inst. Signal Inf. Process. **18**, 1–8 (1998)
4. R. Basri, D.W. Jacobs, Lambertian reflectance and linear subspaces. IEEE Trans. Pattern Anal. Mach. Intell. **25**(2), 218–233 (2003)
5. V. Blanz, T. Vetter et al., A morphable model for the synthesis of 3d faces, in *SIGGRAPH '99* (1999), pp. 187–194
6. P. Borodin, G. Zachmann, R. Klein, Consistent normal orientation for polygonal meshes, in *Proceedings Computer Graphics International, 2004* (IEEE, Piscataway, 2004), pp. 18–25
7. A.M. Bronstein, M.M. Bronstein, R. Kimmel, Expression-invariant 3d face recognition, in *International Conference on Audio- and video-based Biometric Person Authentication* (Springer, Berlin, 2003), pp. 62–70
8. T. Carneiro, R.V.M. Da Nóbrega, T. Nepomuceno, G.-B. Bian, V.H.C. De Albuquerque, P.P. Reboucas Filho, Performance analysis of Google Colaboratory as a tool for accelerating deep learning applications. IEEE Access **6**, 61677–61685 (2018)
9. E. Cengil, A. Çinar, Comparison of Hog (histogram of oriented gradients) and Haar Cascade algorithms with a convolutional neural network based face detection approach. Comput. Sci. **3**(5), 244–255 (2017)
10. C. Cortes, V. Vapnik, Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
11. Q.-K. Do, A. Allauzen, F. Yvon, Modèles de langue neuronaux: une comparaison de plusieurs stratégies d'apprentissage, in *Actes de la 21e conférence sur le traitement automatique des langues naturelles (TALN)* (2014), pp. 256–267
12. Y. Feng, F. Wu, X. Shao, Y. Wang, X. Zhou, Joint 3d face reconstruction and dense alignment with position map regression network, in *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 534–551
13. A.S. Georghiades, P.N. Belhumeur, D.J. Kriegman, From few to many: illumination cone models for face recognition under variable lighting and pose. IEEE Trans. Pattern Anal. Mach. Intell. **34**(6), 643–660 (2001)
14. G. Ghiasi, T.-Y. Lin, Q.V. Le, DropBlock: a regularization method for convolutional networks, in *Advances in Neural Information Processing Systems* (2018), pp. 10727–10737
15. X. Guo, J. Xiao, Y. Wang, A survey on algorithms of hole filling in 3d surface reconstruction. Vis. Comput. **34**(1), 93–103 (2018)
16. G.B. Huang, M. Mattar, T. Berg, E. Learned-Miller, Labeled faces in the wild: a database for studying face recognition in unconstrained environments, in *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition* (2008)
17. S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift (2015). Preprint. arXiv: 1502.03167
18. S. Jahanbin, H. Choi, Y. Liu, A.C. Bovik, Three dimensional face recognition using iso-geodesic and iso-depth curves, in *2008 IEEE Second International Conference on Biometrics: Theory, Applications and Systems* (IEEE, Piscataway, 2008), pp. 1–6
19. D. Jiang, Y. Hu, S. Yan, L. Zhang, H. Zhang, W. Gao, Efficient 3d reconstruction for face recognition. Pattern Recogn. **38**(6), 787–798 (2005)
20. I. Jolliffe, *Principal Component Analysis* (Springer, New York, 2011)
21. D.E. King, Dlib-ml: a machine learning toolkit. J. Mach. Learn. Res. **10**(Jul), 1755–1758 (2009)

22. A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
23. J. Lawrence, J. Malmsten, A. Rybka, D.A. Sabol, K. Triplin, Comparing TensorFlow deep learning performance using CPUs, GPUs, local PCs and cloud, in *Proceedings of Student-Faculty Research Day, CSIS*, Pace University (2017)
24. E. Learned-Miller, G.B. Huang, A. RoyChowdhury, H. Li, G. Hua, Labeled faces in the wild: a survey, in *Advances in Face Detection and Facial Image Analysis* (Springer, Cham, 2016), pp. 189–248
25. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. Nature **521**(7553), 436 (2015)
26. S.P. Lim, H. Haron, Surface reconstruction techniques: a review. Artif. Intell. Rev. **42**(1), 59–78 (2014)
27. X. Liu, M. Kan, W. Wu, S. Shan, X. Chen, VIPLFaceNet: an open source deep face recognition SDK. Front. Comput. Sci. **11**(2), 208–218 (2017)
28. S.J. Pan, Q. Yang, A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2009)
29. O.M. Parkhi, A. Vedaldi, A. Zisserman et al., Deep face recognition, in *BMVC*, vol. 1 (2015), p. 6
30. P. Paysan, R. Knothe, B. Amberg, S. Romdhani, T. Vetter, A 3d face model for pose and illumination invariant face recognition, in *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance* (IEEE, Piscataway, 2009), pp. 296–301
31. J. Roth, Y. Tong, X. Liu, Unconstrained 3d face reconstruction, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 2606–2615
32. S.R. Safavian, D. Landgrebe, A survey of decision tree classifier methodology. IEEE Trans. Syst. Man Cybern. **21**(3), 660–674 (1991)
33. F.S. Samaria, Face recognition using hidden Markov models. PhD thesis, University of Cambridge, Cambridge, UK, 1994
34. F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: a unified embedding for face recognition and clustering, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 815–823
35. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition (2015). Preprint. arXiv: 1409.1556v6
36. L. Sixt, B. Wild, T. Landgraf, RenderGAN: generating realistic labeled data. Front. Robot. AI **5**, 66 (2018)
37. Y. Sun, Y. Chen, X. Wang, X. Tang, Deep learning face representation by joint identification-verification, in *Advances in Neural Information Processing Systems* (2014), pp. 1988–1996
38. Y. Sun, X. Wang, X. Tang, Deep learning face representation from predicting 10,000 classes, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 1891–1898
39. Y. Sun, D. Liang, X. Wang, X. Tang, Deepid3: face recognition with very deep neural networks (2015). Preprint. arXiv: 1502.00873
40. Y. Sun, X. Wang, X. Tang, Deeply learned face representations are sparse, selective, and robust, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 2892–2900
41. Y. Taigman, M. Yang, M. Ranzato, L. Wolf, DeepFace: closing the gap to human-level performance in face verification, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 1701–1708
42. J. Wang, L. Yin, X. Wei, Y. Sun, 3d facial expression recognition based on primitive surface feature distribution, in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2 (IEEE, Piscataway, 2006), pp. 1399–1406
43. K.Q. Weinberger, J. Blitzer, L.K. Saul, Distance metric learning for large margin nearest neighbor classification, in *Advances in Neural Information Processing Systems* (2006), pp. 1473–1480
44. M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in *European Conference on Computer Vision* (Springer, Cham, 2014), pp. 818–833

Improved Plant Species Identification Using Convolutional Neural Networks with Transfer Learning and Test Time Augmentation

Kelvin Igbineweka, Babatunde Sawyerr, and Ebun Fasina

1 Introduction

Machine learning is rapidly gaining popularity in several application areas, ranging from e-commerce recommendation engines to content filtering on social media to intelligent web searches to facial verification and identification. These advancements have been made possible by a special class of techniques known as deep learning [1]. Deep learning can learn complex features using a large training dataset with the help of the so-called backpropagation algorithm [2]. Over the years, these techniques have drastically enhanced the present state of the art in object detection, image and speech recognition, self-driving cars, and several other domains such as genomics, plant classification, and drug discovery. There are several deep learning algorithms for solving various challenges in machine learning. For example, recurrent neural networks (RNNs) are popularly used to model sequential data such as speech and text, while CNNs are widely used for image recognition, object detection, and image processing. Although CNNs have been around since the 1980s [3], their widespread adoption for large-scale image classification tasks began in 2012 when the AlexNet CNN won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [4]. Since then CNNs have dominated the computer vision space, winning most competitions involving image recognition [5].

CNNs are able to take advantage of local spatial characteristics that are found in images, thus making them well suited for image classification tasks. Furthermore, they provide the necessary machinery for learning the features in images, by stacking successive computational layers that provide successive re-representations

K. Igbineweka (✉) · B. Sawyerr · E. Fasina
Department of Computer Science, University of Lagos, Lagos, Nigeria

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_8

of the image. The early layers learn basic features and properties like edges while the later layers detect intricate features like the entire shape of the image. CNNs also use the concept of local connectivity and weight sharing to manage the parameters of the network, thus allowing them to perform well on large datasets.

In this chapter, we apply convolutional neural networks for the identification of plant species from leaves. Plant classification is essential in the study of plants. However, using conventional keys for the identification of plants not only is time consuming but also poses special challenges for two reasons. First, the repertoire of plant species classes is extensive [6]; and second, the taxonomic attributes of plants are highly complex, so that botanists resort to technical terms that are incomprehensible to nonexperts. These challenges pose a formidable barrier to novices who may be interested in obtaining species knowledge.

Existing algorithms for plant species identification use queries relating to the properties of the given plant (e.g., color, number of petals, the presence of thorns or hairs, shape). The answers to these queries are then summed up into a feature vector and sent to a classifier for identification [7]. This task of manually extracting features not only is difficult but also requires specialized domain knowledge and botanical expertise. CNNs solve this problem by being able to automatically detect the feature representation and carry out the needed identification. In this chapter, we propose a convolutional neural network architecture that utilizes the concept of transfer learning for plant species identification. Furthermore, we show that the application of different augmentation techniques can greatly improve the classification accuracy without needing to rely on handcrafted features.

2 Convolutional Neural Networks

A convolutional neural network (ConvNet, or CNN) is a variant of the feed-forward artificial neural network that uses convolution operations to master high-level features in data. In humans, the visual cortex plays the vital role of receiving, integrating, and processing the visual information received from the retinas. There are regions in the visual cortex known as *receptive fields* that contain neurons that emit different responses based on the area of the region that is triggered. In these receptive fields, two classes of neurons can be found: those that fire when low-level features such as edges and dots are detected while there are others that fire when high-level features are detected. This organizational structure is a major inspiration for convolutional neural networks. In a CNN, there are three types of layers: convolution, pooling, and classification. These layer types are described in more detail in the following paragraphs.

Convolutional layers act as feature extractors in the network. Their primary role is to extract from the input relevant features by learning the representation of these features. It achieves this using a set of convolution operation which operates by computing the dot product between the receptive fields and weights locally connected to them [8].

Pooling layers are used to shrink the dimensions of the resulting feature maps. They also play a vital role of introducing spatial invariance where the inputs suffer from some form of distortions and translations [9]. In the past, average pooling was commonly used to calculate and propagate the average values within a receptive field to the succeeding layer [3, 9, 10]. However, in recent models [4, 11–15], max pooling is extensively used to propagate the maximum values within a receptive to the succeeding layer.

The classification layer is the last layer in a CNN and may consist of one or more fully connected layers. It is in this layer that functions relating to high-level reasoning are carried out and the features from the previous layers are represented [11, 14, 16].

3 CNN Architectures

Neural networks usually require very huge computational setup which made deep learning very hard to use up until the late 1990s. One of the first major applications of CNN was for the recognition of handwritten zip codes as proposed by [3]. In this research, the famous MNIST dataset was trained to achieve a 99.7% accuracy. Their network had only convolutional and max pooling layers. Krizhevsky et al. [4] won the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), attaining 15.3% error rate with "AlexNet." While the AlexNet architecture resembles LeNet's, it is bigger in all respects (more layers, more feature maps per layer, more neurons per feature map) and includes stacked convolutional networks: before AlexNet, it was common to have convolutional layers immediately followed by pooling layers to reduce the number of inputs to the next layer. To train AlexNet, a regularization technique called "dropout" was developed to reduce overfitting in the fully connected layers. Training time was reduced through the use of graphics processing units (GPU) to perform a parallel implementation of the convolution operation. Zeiler & Fergus [14] improved on the AlexNet architecture by adjusting the parameters of the network and won the 2013 ImageNet competition, achieving an 11% classification error.

It is hard to train neural networks that are several layers deep due to slowed-down convergence of the training process [17]. This slowing down arises because during backpropagation the weights are adjusted in proportion to the partial derivative of the cost function, and the size of the gradients tends to reduce as the backpropagation reaches further back into the network. Several techniques have been developed to facilitate training of intermediate layers in deep networks, as described in the following paragraphs.

The 2014 ImageNet competition was won by the Google team [12, 13], with a 6% classification error. Their new architecture (code named GoogLeNet) included a new type of layer dubbed "inception module" that uses multiple convolutions of different sizes within a single network to give a "network within network" type structure. This innovation leads to better performance with fewer layers (thus alleviating the gradient-vanishing problem alluded to in the previous paragraph). Since inception modules are more effective in modeling local features, GoogLeNet did not require fully connected final layers as in previous architecture. Instead, based on the recommendation of [18], the fully connected final layer was replaced with max pooling, which is easier to interpret and less prone to overfitting. The researchers found that this replacement led to a 0.6% reduction in classification error. The second place finisher in 2014 with 7% classification error was VGGNet, which lacked inception modules but was deeper than GoogLeNet [11].

Batch normalization (BN) was introduced by [19] as a highly effective technique for speeding up training while enhancing performance. In virtually all neural networks, input data is normalized by rescaling and shifting to put the data within a standard range so that network elements can discriminate efficiently. BN extends this idea by performing a normalization operation between inner layers as well as the before the input layer. Training is done using minibatches, and normalization for each unit in each layer is calculated minibatch by minibatch. Each normalized result is then shifted and scaled using parameters that are trained during the training process. Once training is complete, the minibatch normalization step is replaced by a normalization calculated based on multiple minibatches. This procedure reduces the effect of shifts of inner-layer inputs during the training (these are known as "internal covariate shifts"), and has been found experimentally to reduce training effort by more than an order of magnitude.

Some studies have shown that networks can be made easier to train by including some shortened connections between layers nearest to the input and those nearest to the output [20]. A similar technique was used in ResNet, an ultra-deep model consisting of 152 layers which won ILSVRC 2015 [5]. In ResNet, additional inputs are provided to each layer consisting of outputs from earlier layers that are transformed and summed. Although these networks were very deep, the fact that errors can be channeled directly to units in earlier layers made them easy to train and optimize.

DenseNet [21] realizes similar advantages by using the feature maps from the current layer as well as those from preceding layers as inputs to subsequent layers. The DenseNet architecture includes several "dense blocks," which are blocks of layers in which all layers' outputs are included as inputs to all succeeding layers, as shown in Fig. 1.

This design guarantees better parameter efficiency (i.e., more performance with fewer parameters) and facilitates backpropagation, making DenseNet easier to train.

DenseNet possesses other features to improve computational efficiency and to prevent overfitting when training datasets with smaller sizes. "Bottleneck" layers are inserted between successive layers in the dense block to cut down the number of inputs to the ensuing layer. These bottleneck layers include a batch normalization, a ReLU rectification, and a convolution that combines feature maps. Additionally, between two successive dense blocks, DenseNet places a transition layer (e.g., the final layer shown in Fig. 1), which also reduces the number of layer inputs to the following dense block by a constant fraction.

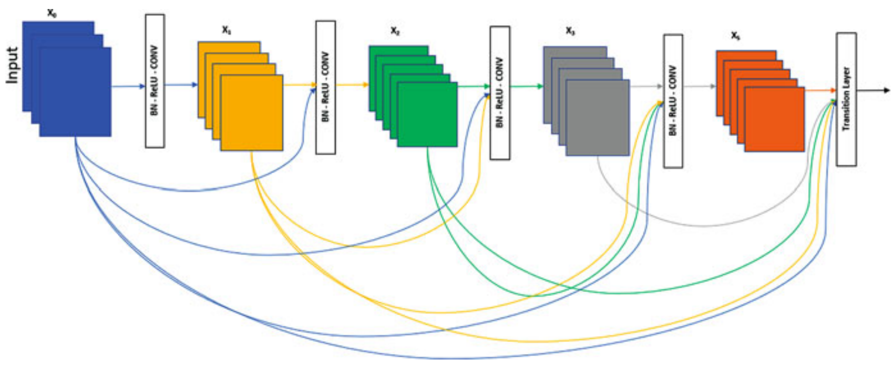


Fig. 1 A dense block with five layers. Each feature map is fed as input to subsequent layers

In practice, CNNs are rarely trained from scratch, since it is often hard to find datasets of sufficient size. A more common approach to training is transfer learning: a network gains "knowledge" by training on one dataset, and is then applied to recognize different but related data. For example, the knowledge gathered from recognizing cars can equally be applied to some extent in recognizing trucks. One variant of the transfer learning concept is fine-tuning [22]. To achieve fine-tuning, we start by training a convolutional neural network (CNN) to learn features from a wide domain with a classification function aimed at reducing the error in that domain. We then replace this function and optimize the network to reduce the error in another, more specific domain. Under this type of learning, we are transferring the parameters and features of the network from the broader domain to the more explicit one.

One final technique for performance enhancement that is simple to implement involves combining the results from multiple models are used to obtain a combined estimate (for example, by majority vote). This technique, known as "ensemble," is often used in deep learning competitions such as Kaggle (kaggle.com/competitions) to give the winner a slight performance boost that edges it past the other entries.

4 Experimental Setup

The dataset used for this research was obtained from the Kaggle web site [23]. Separate datasets were supplied for training and testing. The training dataset consists of 3803 RGB images from 960 distinct plants belonging to 12 species at different growth stages. The images were assembled by [24], and represent some of the most common weed species in Danish agriculture. Since there are so many images per species, any algorithm that correctly classifies them should be able to cope with high intra-class variations. Images from the dataset were randomly separated into training and validation sets, with 70% for training and 30%

for validation. The training dataset was augmented through randomized geometric transformations on the images including flipping, cropping and rescaling, rotation and elastic deformation [25]. Additional augmentation was achieved by randomized transformations of images' brightness through scaling and shifting.

For this experiment, several preprocessing steps were carried out. For normalization, the pixels were first scaled to values between 0 and 1 and then the mean and standard deviation from the pretrained models were applied (normalization has been shown to produce faster convergence [26]).

Three CNNs were used in the experiment: DenseNet161, DenseNet201, and ReNet152. All networks were supplied by PyTorch [27], which is an open-source deep learning library built on top of the Torch library [27]. PyTorch is actively developed by the artificial intelligence research group at Facebook. With PyTorch, it is easy to carry out machine learning computations using tensors. PyTorch also has integrated CUDA® commands that can be used to implement parallel processing on GPUs (Graphics Processing Units). PyTorch is widely used in several deep learning applications, such as natural language processing and computer vision. All computations were performed on the Google Cloud Platform (GCP).

The three networks were all fine-tuned with the same training procedure, described as follows. First, the final layer in each pretrained network (a softmax classifier) was replaced with a new softmax layer that outputs 12 classes and is initialized with random input weights. This classifier was then trained from scratch, using the training data derived from the plant seedling database.

The cutout regularization technique was used to prevent overfitting [28]. Cutout consists of randomly mask out square regions on input during training. For this experiment, a randomly located mask of size 50×50 is used to zero out a square patch of pixels in each input image for every epoch. Specifically, and not all parts of the cutout mask were contained in the image, thus allowing the network to receive some examples where a huge portion of the image is noticeable during training.

The learning rates for different layers were assigned different values. The newly introduced softmax classifier layer was assigned a larger learning rate, since it was initialized with random values which do not reflect any useful information. For the remaining layers, the learning rate was kept small in order to maintain the parameters of the original network and gradually migrate previous knowledge to the new network. These learning rates were further reduced as the training progressed. For this experiment, for the first training epoch a learning rate of 0.1 was used for the newly added softmax layer, while other network parameters were kept fixed (corresponding to learning rate 0). In the second training epoch, the learning rate was set to 0.0003 for all network parameters. As the training progressed, if the log loss failed to improve in two consecutive epochs, then the learning rate is multiplied by 0.1. All models were trained for 15 epochs with the PyTorch implementation of the Adam algorithm [29] used for optimizing the parameters of the network.

For testing, two different strategies were used. The first was to test the images with no augmentation applied. The second method used test time augmentation (TTA) which has been shown to significantly improve the test results of convolutional neural network algorithms [13, 20]. TTA is simply the application of several different transformations to the test images such as flipping, rotation, and translations. These transformed images are then fed to the trained model with the results averaged to get a more reliable prediction. For this experiment, a 12-TTA strategy was used, as follows. First, we predict with the image and five other crops of the image, i.e., top-left, top-right, bottom-left, bottom-right, and center. We do the same thing with the image flipped horizontally and another five crops of the flipped image. The 12 resulting predictions were averaged to give the final prediction, which was used by the Adam algorithm to adjust the weights.

5 Results and Discussion

Training and validation results for the three different architectures are shown in Fig. 2. For DenseNet161, training and validation results converged around epoch 8, and continued to decrease together until epoch 15. For DenseNet201 and ResNet152, training and validation converged around epoch 5, followed by subsequent decreases in both training and validation losses.

Table 1 shows results for the individual models without and with TTA. Both with and without TTA, DenseNet161 was the lowest performer, while ResNet152 was the (matched or unmatched) best performer. TTA gave improvements of between

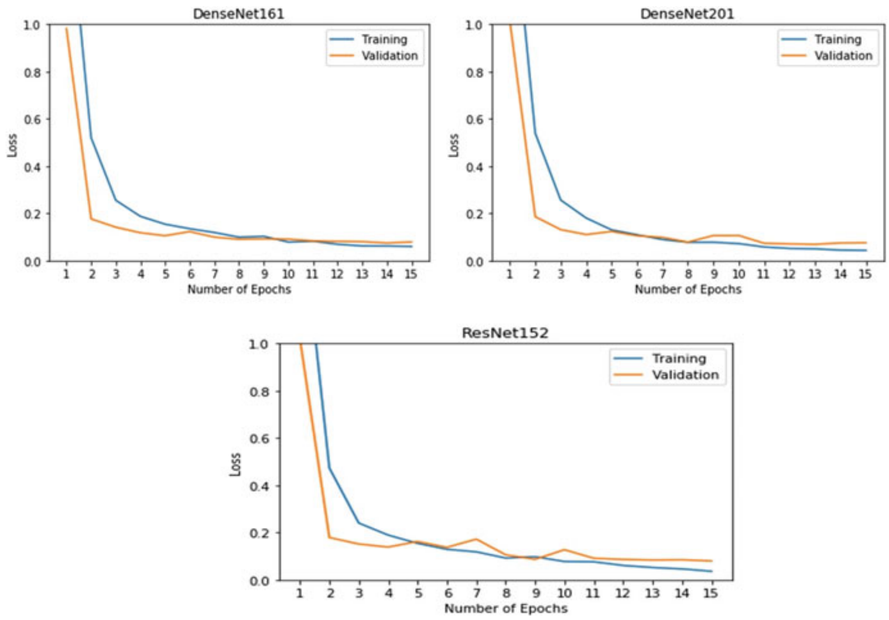


Fig. 2 Training/validation loss vs. number of epochs for DenseNet161, DenseNet201, and ResNet152

Table 1 Results from testing without and with augmentation

Model	Without TTA		With TTA	
	Error rate	Accuracy (%)	Error rate	Accuracy (%)
DenseNet161	0.024287	97.57	0.022175	97.78
DenseNet201	0.020063	97.99	0.020063	97.99
ResNet152	0.020063	97.99	0.016895	98.31

Table 2 Result from testing with ensemble

Method	Error	Accuracy (%)
Ensemble without augmentation	0.021119	97.89%
Ensemble with augmentation	0.014784	98.52

0 and 0.32 percent, with ResNet152 achieving the largest improvement. Overall, ResNet152 with TTA was the best performer, 0.32 percent better than DenseNet201 (with or without TTA).

Table 2 summarizes the results for the ensemble system, which averaged the results of the three baseline networks. When tested without TTA, the accuracy of the ensemble system fell 0.1% short of that achieved by DenseNet201 and ResNet152. On the other hand, ensemble with TTA surpassed ResNet152 by 0.22%, and gave the best overall result.

6 Summary

In this chapter, we have presented a high-performance plant seedling classification algorithm that uses CNN. This algorithm combines three models pretrained on the ImageNet dataset, and was fine-tuned on a much smaller set of 3803 plant images [24] which was augmented using various transformations. The use of test time augmentation (TTA) during testing reduced error rate by 30%, from 0.021 to 0.015. This research has shown that a good classification result can be achieved using relatively few training examples.

References

1. Y. Lecun, Y. Bengio, G. Hinton, Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539
2. P.J. Werbos, Applications of advances in nonlinear sensitivity analysis. Syst. Model Optim, 762–770 (1982). https://doi.org/10.1007/BFb0006203
3. Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**(4), 541–551 (1989). https://doi.org/10.1162/neco.1989.1.4.541
4. A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks. Adv. Neural Inf. Proces. Syst. **25**, 097–1105 (2012). https://doi.org/10.1016/j.compmedimag.2014.06.005

5. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
6. E. Aptoula, B. Yanikoglu, Morphological features for leaf based plant recognition, in *2013 IEEE International Conference on Image Processing, ICIP 2013—Proceedings*, (2013), pp. 1496–1499. https://doi.org/10.1109/ICIP.2013.6738307
7. J. Wäldchen, P. Mäder, Plant species identification using computer vision techniques: a systematic literature review. Arch. Comput. Methods Eng. **25**(2), 507–543 (2018). https://doi.org/10.1007/s11831-016-9206-z
8. W. Rawat, Z. Wang, Deep convolutional neural networks for image classification: A comprehensive review. Neural Comput. **29**(9), 2352–2449 (2017). https://doi.org/10.1162/NECO_a_00990
9. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2323 (1998). https://doi.org/10.1109/5.726791
10. Y. LeCun, B. Boser, J.S. Denker, R.E. Howard, W. Habbard, L.D. Jackel, D. Henderson, Handwritten digit recognition with a back-propagation network. Dermatol. Surg. **39**(1pt2), 149 (1989). https://doi.org/10.1111/dsu.12130
11. K. Simonyan, & A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition (2014), 1–14. doi:https://doi.org/10.1016/j.infsof.2008.09.005
12. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Going deeper with convolutions. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 7–12 June (2015), 1–9. doi:https://doi.org/10.1109/CVPR.2015.7298594
13. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, & Z. Wojna, Rethinking the Inception Architecture for Computer Vision (2015). doi:https://doi.org/10.1109/CVPR.2016.308
14. M.D. Zeiler, & R. Fergus, Visualizing and understanding convolutional networks. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8689 LNCS (Part 1), 818–833 (2014). doi:https://doi.org/10.1007/978-3-319-10590-1_53
15. M.D. Zeiler, & R. Fergus, Visualizing and Understanding Convolutional Networks arXiv:1311.2901v3 [cs.CV] 28 Nov 2013. *Computer Vision–ECCV 2014*, *8689*, 818–833 (2014). doi:https://doi.org/10.1007/978-3-319-10590-1_53
16. G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors. IEEE Signal Process. Mag. **29**(6), 82–97 (2012). https://doi.org/10.1109/MSP.2012.2205597
17. Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(157), 166 (1994)
18. M. Lin, Q. Chen, & S. Yan, Network In Network (2013) p 1–10. doi:https://doi.org/10.1109/ASRU.2015.7404828
19. S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift Sergey. ArXiv **36**(10), 800–805 (2015). https://doi.org/10.1007/s13398-014-0173-7.2
20. K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in *Proceedings of the IEEE International Conference on Computer Vision, 2015 Inter*, (2015), pp. 1026–1034. https://doi.org/10.1109/ICCV.2015.123
21. G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in *Proceedings—30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017-Jan*, (2017), pp. 2261–2269. https://doi.org/10.1109/CVPR.2017.243
22. Y. Bengio, A. Courville, Deep learning of representations for unsupervised and transfer learning. Intel. Syst. Ref. Libr **49**(2011), 1–28 (2013). https://doi.org/10.1007/978-3-642-36657-4_1
23. Plant Seedlings Classification (n.d.) Retrieved August 25, 2019, from https://www.kaggle.com/c/plant-seedlings-classification

24. T.M. Giselsson, R.N. Jørgensen, Jensen, P. K., M. Dyrmann, & H.S. Midtiby, A Public Image Database for Benchmark of Plant Seedling Classification Algorithms (2017). Retrieved from http://arxiv.org/abs/1711.05458
25. O. Ronneberger, P. Fischer, & T. Brox U-net: Convolutional networks for biomedical image segmentation. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9351, 234–241 (2015). doi:https://doi.org/10.1007/978-3-319-24574-4_28
26. Y. LeCun, L. Bottou, G.B. Orr, K.R. Muller, Efficient BackProp. http://Yann.Lecun.Com/Exdb/Publis/Pdf/Lecun-98B.Pdf, *91*(1998), 399–404 (2017)
27. PyTorch (n.d.) Retrieved August 25, 2019, from https://pytorch.org/
28. T. DeVries, & G.W. Taylor, Improved Regularization of Convolutional Neural Networks with Cutout (2017). doi:https://doi.org/10.1016/j.neuron.2007.06.026
29. D.P. Kingma, & J. Ba, Adam: A Method for Stochastic Optimization (2014). Retrieved from http://arxiv.org/abs/1412.6980

Simulation of Biological Learning with Spiking Neural Networks

Chukwuka N. Ojiugwo, Abderazek B. Abdallah, and Christopher Thron

1 Introduction

Artificial neural networks (ANNs) are computational models developed for learning tasks, inspired by brain function [1]. These algorithms learn by examples without being programmed with any task-specific rules. Instead, they generate identifying features from the learning (training) data that they process. For example, ANNs can be trained to recognize images of dogs by systematically analyzing sample images manually labeled as "not dog" or "dog" and using the results to identify dogs in other images. In general, ANNs are connected networks of neurons composed of different layers. Each layer receives inputs from neurons in the previous layer (or layers), does some processing, and passes outputs as input signals to neurons in the next layer, which performs the same functions. Although ANNs have achieved success in many fields (classification tasks, signal processing, recognition tasks, security systems, and so on), they are biologically inaccurate and do not closely mimic the operation mechanism of neurons in animal or human brains.

Spiking neural networks (SNNs) are the third-generation version of ANNs that mimic realistic brain function [2]. Unlike conventional ANNs, SNNs use time-varying signals. Inputs to neurons in SNNs consist of series of current spikes, which affect the neuron's membrane potential (defined as the electric voltage across the

C. N. Ojiugwo (✉)
Department of Computer Science, African University of Science and Technology, Abuja, Nigeria
e-mail: cojiugwo@aust.edu.ng

A. B. Abdallah
School of Computer Science and Engineering, The University of Aizu, Aizu, Japan

C. Thron
Department of Science and Mathematics, Texas A&M University-Central Texas, Killeen, TX, USA

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*,
Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_9

membrane). Neurons fire (or activate) when the membrane potential (or voltage) reaches a specific threshold value. When a neuron fires, it generates current spikes that travel to other neurons through synapses, and the process repeats with these target neurons. Typically, in SNNs, a neuron's membrane potential between input spikes is modeled as a system of differential equations [3] that describe the natural decay of the membrane potential.

The first biologically realistic model of a neuron was proposed by Alan Lloyd Hodgkin and Andrew Huxley in 1952. Their model, known as the Hodgkin-Huxley (HH) model [4], describes the response of the neuron's membrane potential to the input current. The model consists of 4 coupled differential equations in the variables I (input current), V_{m} (membrane voltage), and 3 activation channels. The model has 10 biological parameters that can be measured empirically. The model explains the neuron's spiking behavior, but is only a single-neuron model, and does not describe connections or transmission of signals between neurons.

The FitzHugh−Nagumo (FN) model was proposed in 1962 as a simplified version of the HH model [5]. The model has only two equations, but it is still difficult to analyze and simulate. Furthermore, it has no bursting behavior such as observed in real neurons. To remedy this deficiency, Hindmarsh–Rose proposed a model with three equations that explain bursting behavior [6].

All the biologically based models mentioned above share the following features:

- The ability to generate spikes if the membrane potential crosses a threshold.
- A reset value to initialize the membrane potential after firing.
- A refractory period that prevents the neuron from immediately generating subsequent spikes.
- Complicated dynamics that are difficult to characterize exactly.

The complication of these models has led researchers in the area of ANNs to look for simpler models that have similar behavior but are easier to implement and characterize. The earliest behavioral model of this type is the integrate and fire model, initially investigated by Louis Lapicque in 1907 as discussed in [7]. The model compares the behavior of a neuron to a simple circuit with one resistor and one capacitor in parallel. Although it does not model neurons' biochemical mechanisms, it successfully describes the qualitative behavior of neurons. An improved version of integrate and fire is the leaky integrate and fire model, which will be discussed in the next section.

The remainder of the chapter is organized as follows. In Sect. 2, we describe three alternative functional neuron models that are used in SNNs. In Sect. 3, we describe the spike-timing-dependent plasticity (STDP) learning algorithm and an application of STDP to train a SNN to recognize handwritten digits [8]. In Sects. 4 and 5, we discuss available software SNN simulators and hardware implementations, respectively. In Sect. 6, we summarize our work and discuss current and future research as well as possible practical applications.

2 Mathematical Neuron Models

In this section, we describe the equations and behavior of three mathematical models of neurons: the integrate and fire (IF) and leaky integrate and fire (LIF) models mentioned in the previous section as well as a conductance-based model.

2.1 *Integrate and Fire (IF) Model*

IF neurons have three modes of behavior: an accumulation mode, a firing mode, and a refractory mode [9]. During the accumulation mode, the membrane voltage V_m obey the following equation:

$$\frac{\mathrm{d}V_\mathrm{m}}{\mathrm{d}t} = \frac{I(t)}{C_\mathrm{m}}, \tag{1}$$

where $I(t)$ is the input current and C_m is the membrane capacitance (a model parameter). If the input current is nonnegative, the membrane voltage increases with t until it reaches a threshold, V_th (also a model parameter). At this point the neuron's firing mode is activated, producing a delta function spike in the output current, and resetting the voltage V_m to its resting potential, which is taken as $V_\mathrm{m} = 0$. The neuron then enters refractory mode, and the voltage remains at 0 for a refractory period t_ref. After this, the neuron re-enters accumulation mode, and the process repeats.

Figure 1 shows an SNN neuron's response to the constant input current. As shown in the figure, the period between activations is given by $\frac{CV_\mathrm{th}}{I} + t_\mathrm{ref}$.

Figure 2 shows the IF neuron's response to an input current consisting of regularly spaced pulses. Each pulse produces a step in the membrane voltage, as shown in Fig. 2b. When the voltage steps above the threshold, V_th, an output current pulse is produced and the neuron's voltage resets to 0.

One limitation of the IF model is that the neuron's voltage remains constant until a pulse arrives or firing occurs, which does not realistically depict neuronal behavior. This deficiency is addressed by the leaky integrate and fire model described below.

2.2 *Leaky Integrate and Fire (LIF) Model*

The LIF model is similar to the IF model, except an additional "leak" term is added to keep neuron voltage from remaining constant during periods of non-stimulation [10]. Before firing, the LIF neuron membrane voltage, V_m, obeys the following equation:

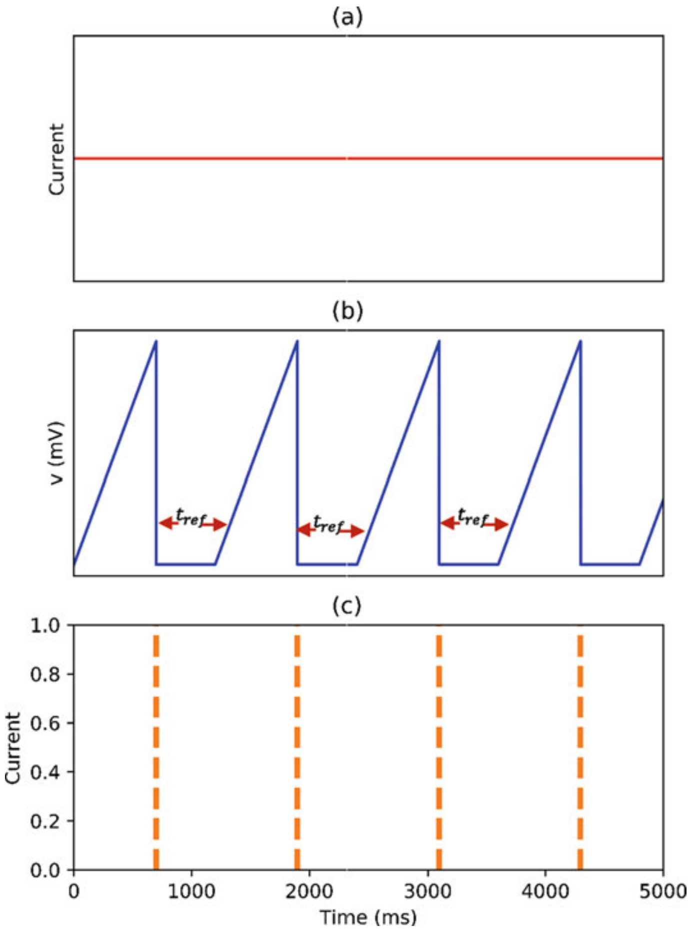


Fig. 1 IF Model with constant input current: (**a**) constant input current, (**b**) neuron voltage (IF-model), (**c**) output current

$$\frac{dV_m}{dt} = \frac{I(t)}{C_m} - \frac{V_m(t)}{C_m R_m}, \tag{2}$$

where R_m is the membrane resistance and $I(t)$ and C_m are the input current and membrane capacitance as before. The factor $\frac{V_m}{R_m}$ in (2) denotes current resulting from the diffusion of ions that occurs via the membrane if some balance is not met in the cell. Activation and refractory modes are the same as in the IF model.

Figure 3 shows a case where the input current is constant. As shown in the figure, the period between activations is given by $t_{act} + t_{ref}$, where $t_{act} = -\text{CR}(\log(\text{RC}) - \log(\text{IR} - V_{th}))$.

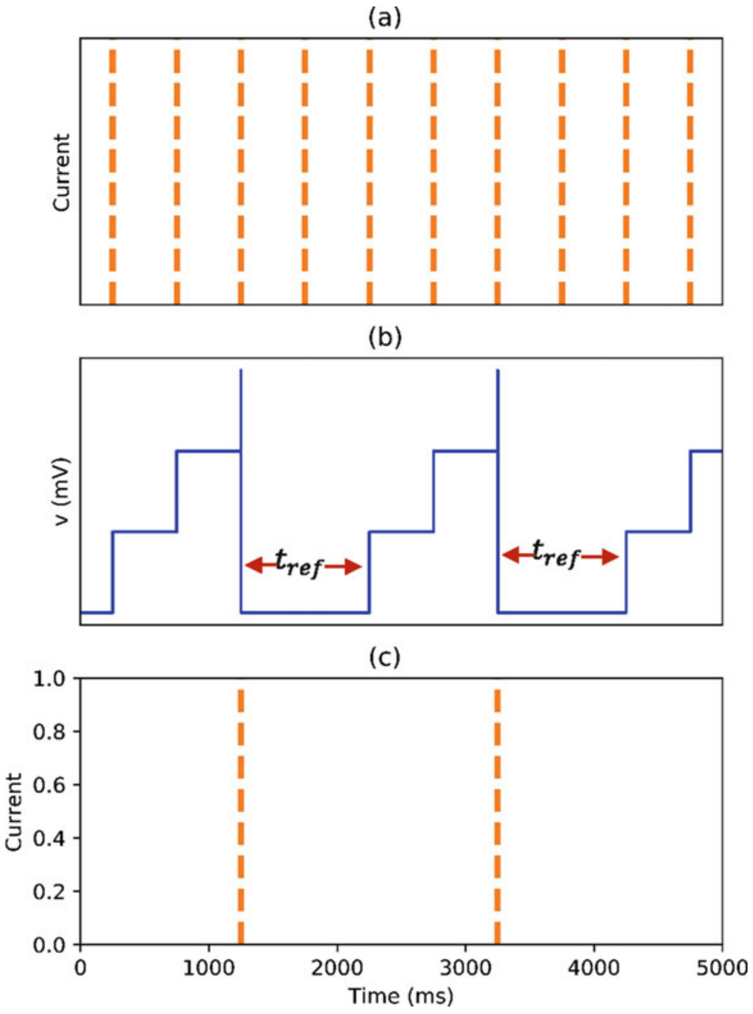


Fig. 2 IF Model with pulse input current (plot was created using the Brian2 software simulator—see Sect. 4.2): (**a**) pulses(spikes) input current, (**b**) neuron voltage (IF-model), (**c**) output current

Figure 4 shows the response of an LIF neuron to a series of input current pulses, and is analogous to Fig. 2 which showed the response of an IF neuron in the same situation. Each pulse produces a step followed by a down-sloped arc in the membrane voltage. When the voltage steps above the threshold, V_{th}, an output pulse is produced and the voltage resets to 0.

In general, neurons are connected to multiple presynaptic neurons which provide inputs. Figure 5 shows an example of an LIF model neuron connected to several input neurons through synapses.

In Fig. 5, three presynaptic neuron inputs (x_1, x_2, x_3) are connected to the neuron through different synapses. Each presynaptic neuron fires at a different rate, and

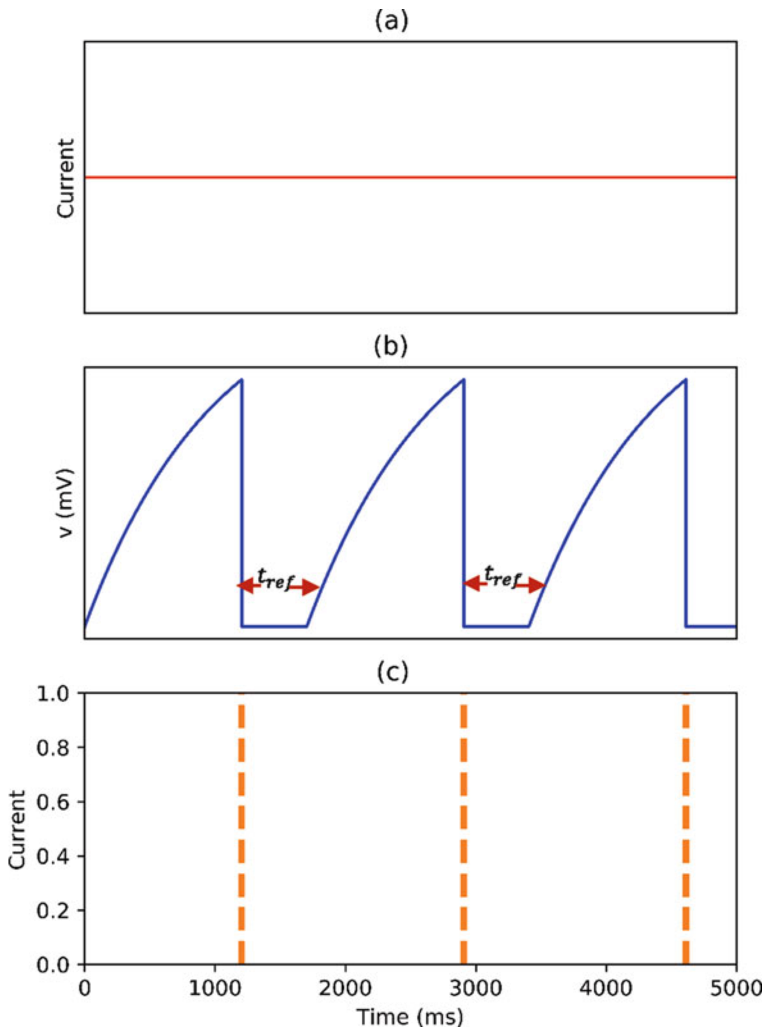


Fig. 3 LIF Neuron Model with constant input current (figures created with Brian2 simulator): (**a**) constant input current, (**b**) neuron voltage (LIF-model), (**c**) output current

each input spike's amplitude depends upon the strength of the connecting synapse. A typical simulation of a neuron's response is shown in Fig. 6. In Fig. 6d, the three input spike trains from (a–c) are combined and weighed according to the weights of the three different connecting synapses. Comparing (d) and (e), we may see that spikes with higher weights produce higher jumps in the neuron voltage. Both the timing and weights of the input spikes are important in determining the voltage response of the neuron, which in turn determines the spiking times of the neuron (see (f)). The relation between synaptic weights, spike timing, and neuron response is

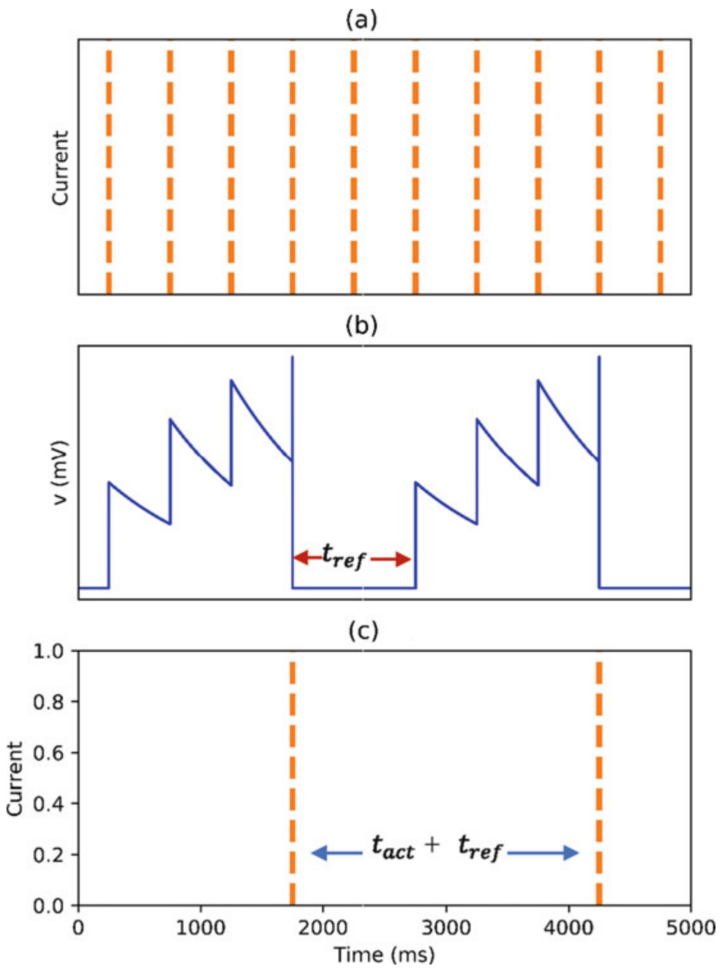


Fig. 4 LIF Neuron Model with pulse input current (figure produced using Brian2 software simulator): (**a**) pulse input current, (**b**) neuron voltage (LIF-model), (**c**) output current

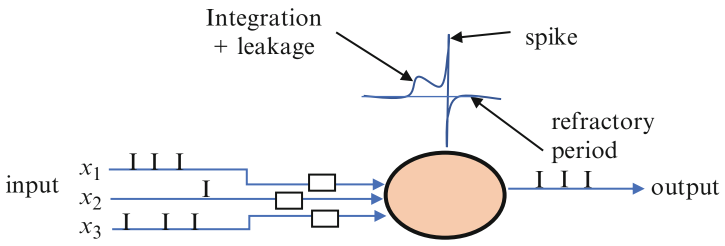


Fig. 5 A simple neuron with three inputs connected through different synapses

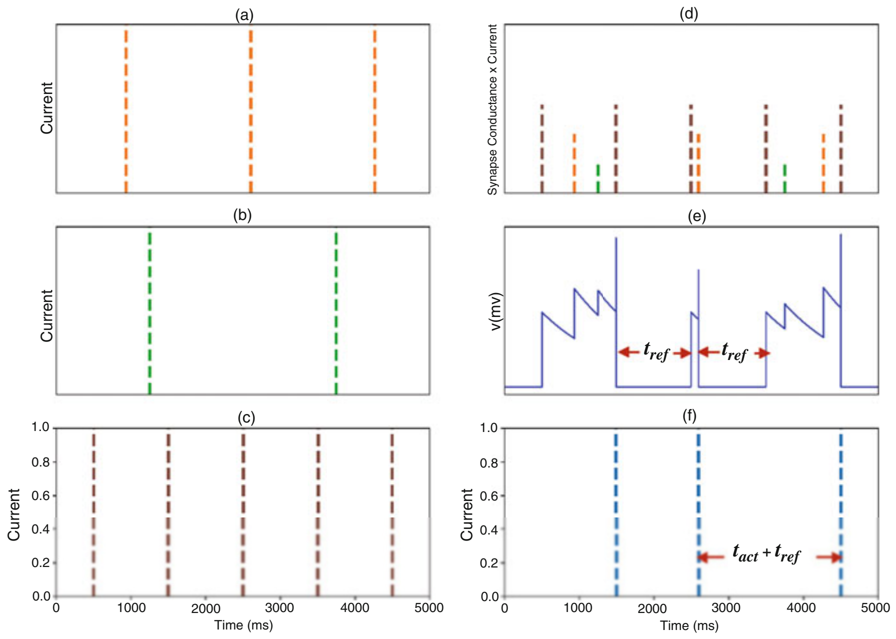


Fig. 6 LIF Neuron Model with multiple input spike(pulse) trains (figure produced using Brian2 software simulator): (**a**) input spike-train 1, (**b**) input spike-train 2, (**c**) input spike-train 3, (**d**) weighted input current from spike trains(1, 2, 3), (**e**) neuron voltage (LIF-model), (**f**) output current

central to the spike-timing-dependent plasticity learning rule that will be discussed in Sect. 3.

In general, the total input current to neuron i is the sum over all current pulses from input neurons that are connected to i via synapses. The mathematical expression for the input current $I_i(t)$ is:

$$I_i(t) = \sum_j w_{ij} \sum_f \delta\left(t - t_{j,n}\right), \tag{3}$$

where $t_{j,\,n}$ represents the time of the nth spike of the jth presynaptic neuron; w_{ij} is the strength (weight) of synaptic efficacy between neurons j and i; and δ represents the Dirac delta function.

2.3 Conductance-Based Neuron Model

The LIF model has several realistic features: input spike trains, synapses with different weights, and neuron voltages that decay with time all reflect observed behavior of actual neurons. However, it differs significantly from the biological-

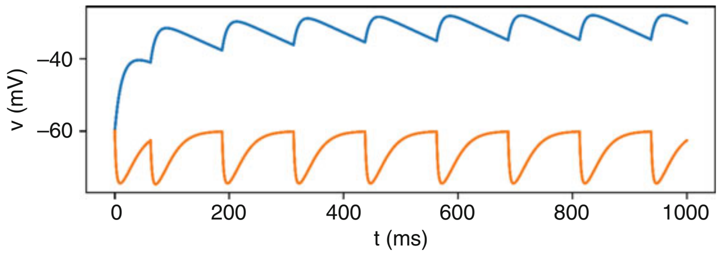


Fig. 7 Conductance based neuron models in Brian2 based on Eq. (4). The figure shows an upper trace of pre excitatory ($g_i = 0$); and a lower trace of pre inhibitory ($g_e = 0$). Note: (figure produced using Brian2 software simulator)

based models (HH, FN, HR models) in which changes in membrane voltage are caused by changes in neuron conductance. A simple conductance-based behavioral model for membrane voltage is given in [8];

$$\tau \frac{\mathrm{d}V}{\mathrm{d}t} = (E_{\text{rest}} - V) + g_e (E_{\text{exc}} - V) + g_i (E_{\text{inh}} - V), \tag{4}$$

where τ is a time constant, E_{rest} defines the membrane resting potential, and E_{exc}, E_{inh}, g_e, and g_i are the equilibrium potentials and conductivities of excitatory and inhibitory synapses, respectively. In this model, the neuron is supposed to be connected to two different types of input neurons: excitatory and inhibitory. When an excitatory input neuron sends a signal to the neuron, the conductance g_e is increased: the amount of increase depends on the weight of the synapse connecting the input neuron to the neuron. Similarly, when an inhibitory neuron signals the neuron, the conductance g_i is increased according to synapse weight. Apart from these increases, the excitatory and inhibitory conductance decay exponentially according to the equations:

$$\tau_{g_e} \frac{\mathrm{d}g_e}{\mathrm{d}t} = -g_e; \qquad \tau_{g_i} \frac{\mathrm{d}g_i}{\mathrm{d}t} = -g_i, \tag{5}$$

Typically, $\tau_{g_e} > \tau_{g_i}$, since it has been observed experimentally that excitatory synapses decay slower than inhibitory synapses.

Once the neuron's membrane potential crosses its threshold V_{thres}, the neuron fires and its membrane potential resets to V_{rest}. As in the IF and LIF models, a refractory period follows during which the neuron's voltage remains at V_{rest}.

In the conductance-based model, the neuron's response is highly dependent on the weights of (excitatory and inhibitory) input neurons. This fact can be made use of to train the neural network by adjusting synaptic weights. In the next section, we will present the spike-timing-dependent-plasticity algorithm, which uses this method for SNN training (Fig. 7).

3 Spike-time-dependent plasticity learning algorithm

In conventional ANNs, the backpropagation algorithm is widely used for training. It is possible to use a version of backpropagation to train SNNs, but this requires recharacterizing the input and output signals in terms of "spike rates," which has no biological basis [11]. An alternative learning algorithm that does not require translation to spike rates is spike-timing-dependent plasticity (STDP) [12]. The STDP learning rule is based on actual neuron behavior, as observed by [13]. They observed that if a neuron fires shortly after an input spike, then the weight of the synapse connecting the input to the neuron tends to increase. On the other hand, if an input neuron spikes shortly after the neuron fires, then the weight of the corresponding synapse decreases. In the usual terminology, input neurons' spikes are referred to as "presynaptic," while the responding neuron's spike is called "postsynaptic."

3.1 Description of STDP

There are several variants of STDP: the following presentation is based on the documentation for the Brian2 software simulator. A mathematical expression for the synapse weight $w(t)$ which reflects the synapse-changing behavior described in [13] is:

$$w(t) = \mathrm{w}_0 + \sum_{t_{\mathrm{pre}}<t} \sum_{t_{\mathrm{post}}<t} W\left(t_{\mathrm{post}} - t_{\mathrm{pre}}\right), \tag{6}$$

where w_0 is the starting weight, t_{pre} and t_{post} represent pre- and postsynaptic spike times, respectively, and $W(s)$ is a fixed function (specified by the model) which is negative when $s < 0$ and positive when $s > 0$. A common form for the function $W(s)$ is:

$$W(s) = \begin{cases} -A_{-}\mathrm{e}^{\frac{-|s|}{\tau_{-}}} & s < 0 \\ A_{+}\mathrm{e}^{\frac{-|s|}{\tau_{+}}} & s > 0 \end{cases}, \tag{7}$$

(see Fig. 8) where A_{-}, $A_{+}> 0$ are model parameters that specify the magnitudes of weight changes, and τ_{-} and τ_{+} denote time decay constants that reflect the steepness of the function for $s < 0$ and $s > 0$, respectively.

To implement Eq. (6), it is best to find simplified equations for the weight change when a new presynaptic or postsynaptic spike arrives. First, a presynaptic spike arriving at time t produces a change Δw in the value of the synaptic weight given by (6), where Δw is given by the function $\mathrm{tr}_{\mathrm{pre}}(t)$ defined as:

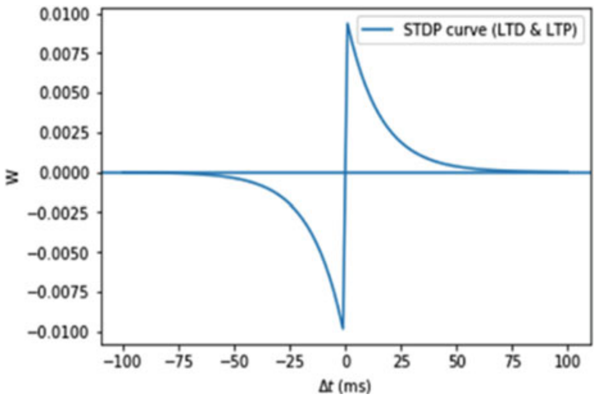


Fig. 8 STDP weight function $W(t)$

$$\varDelta\, w = \mathrm{tr}_{\mathrm{pre}}(t) = \sum_{t_{\mathrm{post}}<t} W\left(t_{\mathrm{post}} - t\right) = \sum_{t_{\mathrm{post}}<t} -A_{-}\mathrm{e}^{\frac{-|t_{\mathrm{post}}-t|}{\tau_{-}}} = \left(-A_{-}\sum_{t_{\mathrm{post}}<t} \mathrm{e}^{\frac{t_{post}}{\tau_{-}}}\right)\mathrm{e}^{\frac{-t}{\tau_{-}}} \tag{8}$$

Following reference [14], we refer to this function as a "trace" because it evolves continuously even when no presynaptic spikes occur. Note that the function $\mathrm{tr}_{\mathrm{pre}}(t)$ only has a practical effect on the system when a presynaptic spike occurs at time t: the trace is only used as a way of keeping track of the previous spiking history.

From the form of Eq. (8), we may see that the function $\mathrm{tr}_{\mathrm{pre}}$ satisfies the differential equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{tr}_{\mathrm{pre}} = -\frac{\mathrm{tr}_{\mathrm{pre}}}{\tau_{\mathrm{pre}}} \tag{9}$$

The arrival of postsynaptic spikes also affects the trace $\mathrm{tr}_{\mathrm{pre}}$. If a postsynaptic spike occurs at time t, then an additional term is introduced into the sum in Eq. (9) which produces a downward jump discontinuity in the trace:

$$\text{Postsynpatic spike at time } t \quad \Rightarrow \quad \mathrm{tr}_{\mathrm{pre}}(t) = \mathrm{tr}_{\mathrm{pre}}\left(t^{-}\right) - A_{-} \quad , \tag{10}$$

where $\mathrm{tr}_{\mathrm{pre}}(t^{-})$ is the value of the trace just before time t (before the jump) and $\mathrm{tr}_{\mathrm{pre}}(t)$ is the new value following the jump.

In summary, when a presynaptic spike occurs, the synaptic weight experiences a change $w \rightarrow w + \mathrm{tr}_{\mathrm{pre}}(t)$, where $\mathrm{tr}_{\mathrm{pre}}(t)$ satisfies Eqs. (6) and (7).

On the other hand, if a postsynaptic spike arrives at time t, it produces a synaptic change Δw given by the function $t_{\mathrm{post}}(t)$, where

$$\mathrm{tr}_{\mathrm{post}}(t) = \sum_{t_{\mathrm{pre}}<t} W\left(t_{\mathrm{pre}} - t\right) = \sum_{t_{\mathrm{pre}}<t} A_{+}\mathrm{e}^{\frac{-|t_{\mathrm{pre}}-t|}{\tau}} = \left(A_{+}\sum_{t_{\mathrm{pre}}<t} \mathrm{e}^{\frac{t_{\mathrm{pre}}}{\tau}}\right)\mathrm{e}^{\frac{-t}{\tau}} \tag{11}$$

From (11) we may obtain a differential equation for $\mathrm{tr}_{\mathrm{post}}$ similar to (9):

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{tr}_{\mathrm{post}} = -\frac{\mathrm{tr}_{\mathrm{post}}}{\tau_{\mathrm{post}}} \tag{12}$$

If a presynaptic spike occurs at time t, then an additional term is also introduced into the sum in Eq. (11) which produces an upward jump in the trace:

$$\text{Presynpatic spike occurs at time } t \quad \Rightarrow \quad \mathrm{tr}_{\mathrm{post}}(t) = \mathrm{tr}_{\mathrm{post}}\left(t^{-}\right) + A_{+} \tag{13}$$

In summary, when a postsynaptic spike occurs, the synaptic weight experiences a change $w \rightarrow w + \mathrm{tr}_{\mathrm{post}}(t)$, where $\mathrm{tr}_{\mathrm{post}}$ satisfies (12) and (13).

3.2 *Handwritten digit recognition using STDP*

Diehl and Cook [8] presented an SNN model that uses unsupervised learning. They used biologically plausible system components, including conductance-based synapses and STDP learning. Figure 9 shows the neuron model architecture which they used. The inputs to the model are images of 28 × 28 pixels. The model comprises two layers: an input layer and a processing layer. The input layer has 28 × 28 neurons, where each neuron is connected to one pixel of the input image. Each input neuron converts its pixel's intensity to a spike train, and the times

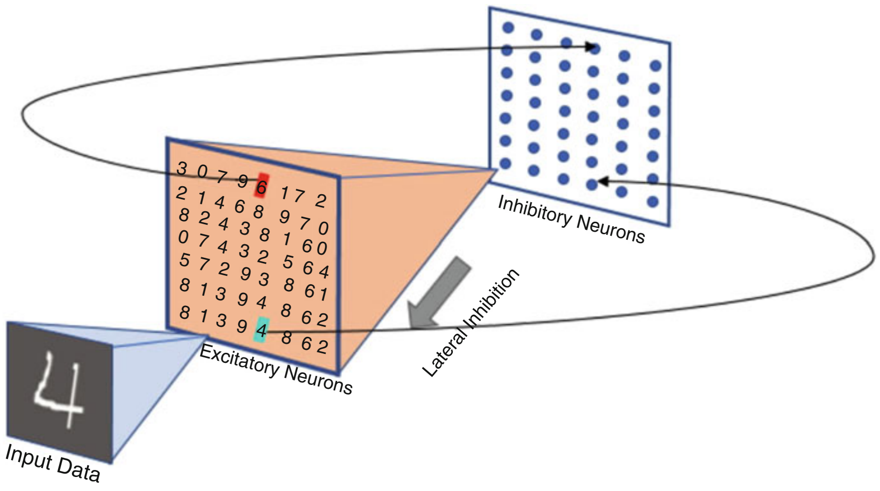


Fig. 9 Network architecture for unsupervised learning of handwritten digits [8]. The figure shows the input layer (labeled "Input Data") and the processing layer consisting of excitatory and inhibitory neurons connected laterally

between spikes are Poisson-distributed and the mean rate of spiking is proportional to the corresponding pixel's intensity.

The processing layer consists of equal numbers of excitatory and inhibitory neurons (different layer sizes were used, as explained below). Each excitatory neuron is connected to all input neurons. Each inhibitory neuron takes input from one excitatory neuron and sends outputs to all other excitatory neurons. The effect of the inhibitory neurons on the excitatory neurons is termed "lateral inhibition."

The membrane voltages for neurons in the processing layer have a conductance-based response to inputs as defined by Eqs. (4) and (5). The outputs of excitatory neurons are modified to prevent any single neuron from dominating the response pattern. Specifically, the firing threshold of the neuron is given by $V_{\mathrm{thresh}} + \theta$, where θ increases by a fixed step every time the neuron fires, and then exponentially decays. This modification of neuron firing is called "homeostasis," and limits the firing rate of highly stimulated neurons so that other neurons have a chance to compete.

All synapses weights from input to excitatory neurons were learned using STDP. For purposes of comparison, three different STDP rules were used. The first rule uses a presynaptic trace with no postsynaptic trace, and the weight change Δw_{post} when a postsynaptic spike occurs is:

$$\Delta w_{\mathrm{post}} = \quad \left(\mathrm{tr}_{\mathrm{pre}} - \mathrm{tr}_{\mathrm{tar}}\right)(w_{\max} - w)^{\mu} \tag{14}$$

where ŋ, $w_{\max}$, and μ determine the learning rate, the maximum weight, and the dependence of the update on the previous weight; and $\mathrm{tr}_{\mathrm{tar}}$ defines the target value of the presynaptic trace when a postsynaptic spike occurs. The third rule uses both pre- and postsynaptic traces with a power-law dependence on weight. No significant difference between the different STDP rules' performances was noted.

The MNIST dataset [15] serves as input to the model. This well-known dataset comprises 60,000 training and 10,000 testing samples of 28 × 28-pixel images of handwritten digits (0–9), respectively. Each of the 576 neurons in the input layer converts one pixel's intensity to a Poisson-distributed spike train input of duration 350 ms. The neuron's mean firing rate was set to be proportional to the pixel intensity: if for any input the excitatory neurons in the second layer fired fewer than five spikes during the 350 ms period, then the constant of proportionality was increased until a minimum of five spikes were fired. During training, MNIST training images were presented with a 150 ms interval between images to allow neuron variables to reset. Once training is completed, the synapse weights and neuron thresholds are fixed, and each neuron is assigned a class (0–9) according to the digit that responded most strongly throughout the course of the training set. The classification accuracy of the model was determined using the test set of 10,000 samples. During testing, the predicted digit is determined by taking the average responses of each neuron per class, and the class with the peak average firing rate gets selected.

Altogether Diehl and Cook trained four models of 100, 400, 1600, and 6400 excitatory neurons (with an equal number of inhibitory neurons), respectively.

Larger models were trained with multiple passes of the MNIST training set: the larger the model, the longer training took to converge. Classification accuracy (as measured with the test set) ranged from 83% with 100 excitatory neurons up to 95% with 6400 excitatory neurons, using the power-law weight-dependent STDP rule. These results demonstrate that high classification accuracy can be achieved using an SNN with biologically realistic parameters for unsupervised learning.

4 SNN Simulation Software

4.1 Overview

Several different SNN software simulators are available. In these simulators, neuron models are implemented in the form of ODEs that represent biological neurons. Simulators can be synchronous (clock-driven) or asynchronous (event-driven), so they run in discrete time or in abstract time, respectively. Generally, the larger the number of neurons simulated, the slower the simulation, since the simulator substrate has a finite computational power shared between all the neurons. For medium scale neural networks, the time relation goes below the real-time boundary, making it slower than the real time.

Figures 1, 2, 3, 4, 5, and 6 above were generated using the Brian2 software simulator. In the following subsections, we describe several SNN software simulators, including Brian2. Properties of these simulators are summarized in Table 1.

4.2 Brian2 Simulator

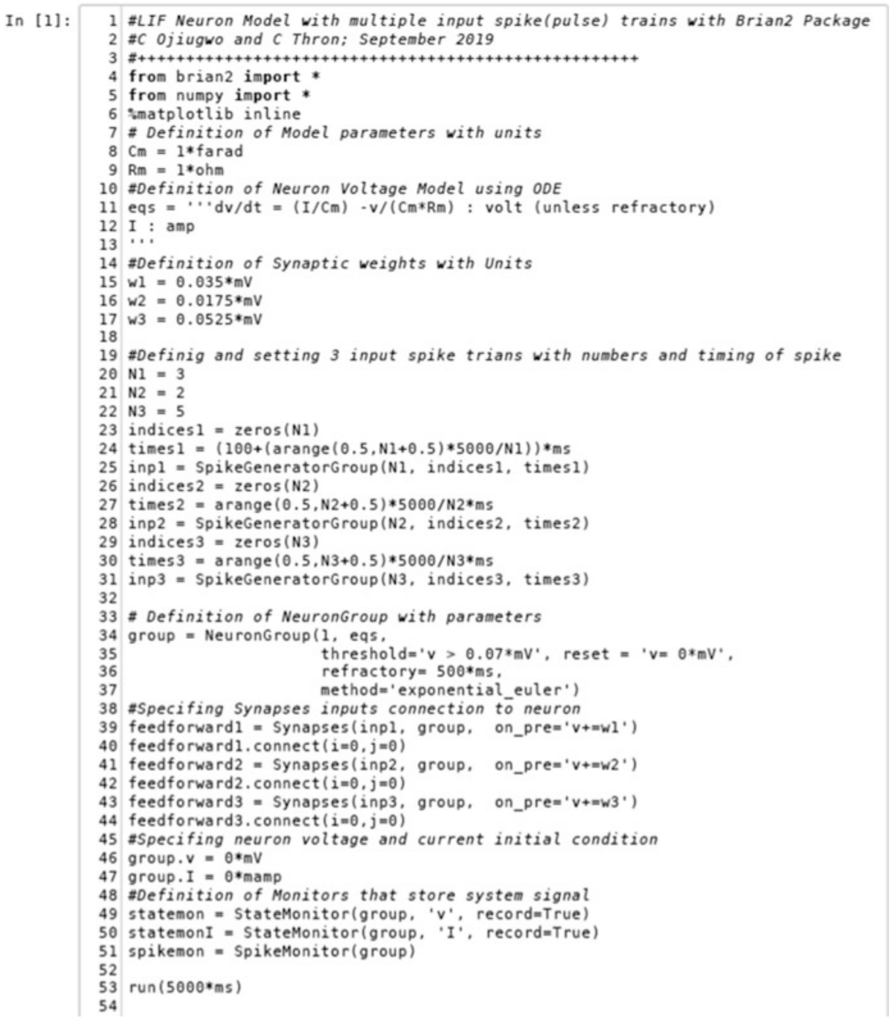
The Brian2 simulator [16] is an open-source Python package for implementing spiking neuron models. The primary aim of Brian2 is to make the writing of simulation code fast and flexible for the developer. It allows developers to spend

Table 1 Comparison of SNN software simulators

Simulator	License	Platform	Language	Focus	Parallelism	PyNN
Brian2	GNU GPL	Linux, Windows, Mac	Python	Neurons Networks	Distributed GPU	Yes
NEURON	GNU GPL	Linux, Unix, OS X, Windows	Fortran, Python, C, C++	Neurons Networks	MPI	Yes
GENESIS	GNU GPL	Linux, Mac, Windows	C	Neurons		No
NEST	GNU GPL	Linux, Unix, OS X	Python, C++, Cython	Neurons Networks	Distributed MPI	Yes

more time on the details of their models, and less on their implementation. Brian2 can implement various neuron models, although it is typically much slower compared to hardware implementation when complex ANNs are involved. Brian2 is clock-driven, and all events occur on a fixed time grid. The Brian2 python code uses vector-based computational techniques for efficient calculations [17]. The simulator is available on Linux, Windows, and Mac operating systems.

Figure 10 gives an example of a Python3 implementation of a neuron model with multiple inputs using the Brian2 package. Model parameters with units are defined in lines 7–9. In lines 10–13 the neuron voltage model is defined using an ODE in standard mathematical form. The model has three input spike trains, connected to



```
In [1]:   1 #LIF Neuron Model with multiple input spike(pulse) trains with Brian2 Package
          2 #C Ojiugwo and C Thron; September 2019
          3 #++++++++++++++++++++++++++++++++++++++++++++++++++++
          4 from brian2 import *
          5 from numpy import *
          6 %matplotlib inline
          7 # Definition of Model parameters with units
          8 Cm = 1*farad
          9 Rm = 1*ohm
         10 #Definition of Neuron Voltage Model using ODE
         11 eqs = '''dv/dt = (I/Cm) -v/(Cm*Rm) : volt (unless refractory)
         12 I : amp
         13 '''
         14 #Definition of Synaptic weights with Units
         15 w1 = 0.035*mV
         16 w2 = 0.0175*mV
         17 w3 = 0.0525*mV
         18 
         19 #Definig and setting 3 input spike trians with numbers and timing of spike
         20 N1 = 3
         21 N2 = 2
         22 N3 = 5
         23 indices1 = zeros(N1)
         24 times1 = (100+(arange(0.5,N1+0.5)*5000/N1))*ms
         25 inp1 = SpikeGeneratorGroup(N1, indices1, times1)
         26 indices2 = zeros(N2)
         27 times2 = arange(0.5,N2+0.5)*5000/N2*ms
         28 inp2 = SpikeGeneratorGroup(N2, indices2, times2)
         29 indices3 = zeros(N3)
         30 times3 = arange(0.5,N3+0.5)*5000/N3*ms
         31 inp3 = SpikeGeneratorGroup(N3, indices3, times3)
         32 
         33 # Definition of NeuronGroup with parameters
         34 group = NeuronGroup(1, eqs,
         35                     threshold='v > 0.07*mV', reset = 'v= 0*mV',
         36                     refractory= 500*ms,
         37                     method='exponential_euler')
         38 #Specifing Synapses inputs connection to neuron
         39 feedforward1 = Synapses(inp1, group,  on_pre='v+=w1')
         40 feedforward1.connect(i=0,j=0)
         41 feedforward2 = Synapses(inp2, group,  on_pre='v+=w2')
         42 feedforward2.connect(i=0,j=0)
         43 feedforward3 = Synapses(inp3, group,  on_pre='v+=w3')
         44 feedforward3.connect(i=0,j=0)
         45 #Specifing neuron voltage and current initial condition
         46 group.v = 0*mV
         47 group.I = 0*mamp
         48 #Definition of Monitors that store system signal
         49 statemon = StateMonitor(group, 'v', record=True)
         50 statemonI = StateMonitor(group, 'I', record=True)
         51 spikemon = SpikeMonitor(group)
         52 
         53 run(5000*ms)
         54 
```

Fig. 10 Example of Brian2 code for simulating a neuron with three spike train inputs

the neuron by 3 synapses. Lines 15–17 give the synapse weights, and lines 20–31 give characteristics of the three input spike trains including number and timing of spikes. Lines 34–37 define a NeuronGroup, which is a group of neurons that share the same equations (models) defining their properties. In this model, the neuron group consists of a single neuron. Lines 39–44 specify the synapses connecting the inputs to the neuron, including the effect of input spikes on the neuron voltage. Lines 46–47 give the neuron voltage and current initial conditions, and lines 49–51 define monitors that store system signals that can be displayed in plots: Fig. 6e, f give the outputs from the neuron voltage state monitor (`statemon`) and spike monitor (`spikemon`), respectively. For more details, readers are directed to the online Brian2 documentation [14]. Also, our Brian2-Python codes can be found on the book's GitHub link: https://github.com/chuks-ojiugwo/Implementations-and-Applications-of-Machine-Learning.

4.3 NEURON Simulator

NEURON simulator is a GNU general public license (GPL) software for modeling neuron models originally developed at Yale and Duke universities [18]. Although initially written in hoc (a C-based programming language), a Python script is also available. The NEURON simulator supports parallelization using the message passing interface (MPI) protocol, which makes it possible to be used on multicore computers. NEURON supports a graphic user interface (GUI) that is available for users with limited programming knowledge, which explains its popularity in computational neuroscience courses and laboratories around the world. With the GUI, it is possible to generate publication-quality results without having to write programming codes at all. It uses both discrete clock-driven and event-driven time paradigm in its operation. Like Brian2, the simulator enables multiple neuron models in the same simulation and equally implements learning features such as STDP. Also, like Brian2, NEURON runs slower than hardware implementations and is available on Linux, Windows, and Mac operating systems.

4.4 GENESIS Simulator

The GEneral NEural SImulation System (GENESIS) simulator developed by James M. Bower at Caltech [19] was the first simulator to enable the construction of large-scale neural networks. The simulator aims to reproduce the biological behavior of neural systems, from the level of biochemical reactions up to large-scale neural networks. GENESIS uses a high-level simulation language to construct neurons and their networks. In practice, GENESIS input commands are entered either by script file, GUI, or GENESIS command shell. The scripting language and the modules are powerful enough that only a few lines of script are needed to specify a sophisticated simulation. At present, GENESIS allows parallelized modeling of single neurons and networks on multiple-instruction-multiple-data parallel computers. Like NEURON and Brian2 simulators, GENESIS allows multiple neuron models in the same simulation, as well as implementing learning features. GENESIS is also slow compared to hardware implementation and is available on Linux, Windows, and Mac operating systems.

4.5 NEST Simulator

NEST is a simulation software for SNN models that support large-scale neuron networks [20]. It is GPL licensed and written in the C++, Python, and Cython programming languages. The NEST simulator provides high accuracy and precision of its simulations. It supports parallel and distributed simulation, using OpenMP or POSIX Threads and the NEST simulation kernel, respectively. PyNEST is the primary NEST user interface that supports Python libraries. NEST supports other simulators, such as PyNN (which also supports Brian2, NEURON, and neuromorphic hardware).

5 Hardware Implementations

5.1 Overview

Hardware implementations of SNNs include brain-inspired processors (also known as neuromorphic chips), and computing systems build with such processors. Neuromorphic chips replicate biological brain functions at the hardware level and use basic analog and digital components to perform these functions much faster than is possible in software simulations [21]. Systems built using these chips can simulate millions of neurons and billions of synapses, and are capable of performing complex real-time cognition tasks. Applications include sensor networks, self-driving automobiles, smartphones, robots, medical imaging, public safety, real-time video analysis, signal processing, olfactory detection, and digital pathology [22]. The following paragraphs introduce some of the most prominent neuromorphic hardware implementations, whose properties are summarized in Table 2.

5.2 IBM TrueNorth

The IBM TrueNorth is a multicore neuromorphic chip designed by IBM solely for SNNs [23]. The chip has 4096 cores, and each core has 256 programmable, fully connected neurons. As an interconnected network of neurosynaptic cores,

Table 2 Comparison of SNN neuromorphic hardware

Name (type)	Developer (year)	Number of cores	Number of neurons	Number of synapses	Power consumption
IBM TrueNorth (processor)	IBM (2014)	4096	4096×256	1.6×10^{10}	70 mW
ROLLS (processor)	ETH Zurich (2015)	Analog (no cores)	256	1.28×10^{5}	Approx. 4 mW
NeuroGrid (system)	Stanford (2009)	16 Neurocores	$16 \times 256 \times 256$	6×10^{6}	3 W
SpiNNaker (system)	U. Manchester (2019)	1,036,800 ARM9 cores	1000 neurons per core	>1000 per neuron	90 kW

TrueNorth has an architecture completely unlike conventional microprocessors. TrueNorth is event-driven and highly energy-efficient, consuming 45 mW for a million synapses, over 2000 times less than the current generation of conventional computer chips. The TrueNorth chip implements "gray matter" on a spike-based messaging network with an intracore crossbar memory, and "white matter" using long-range connections. According to the research work of [23], a TrueNorth neuron can replicate 20 different biological neuron functions and behaviors. The TrueNorth design and architecture supports cognitive computing [24], and has been used to classify video images at more than 1000 frames per second [25]. It has also been used in robots to enable them to navigate difficult terrain [26].

5.3 Reconfigurable On-Line Learning Spiking (ROLLS) neuromorphic processor

The Reconfigurable On-Line Learning Spiking (ROLLS) neuromorphic processor is a complete custom mixed-signal implementation, developed at ETH [27]. ROLLS mimics the functions of biological spiking neurons and synapses for exploring neuroscience models and building brain-inspired computing systems. By supporting a wide range of cortical-like computational modules composed of plasticity mechanisms, this device enables the realization of intelligent autonomous systems with online learning capabilities. The design is robust and capable enough to run a wide range of activities like recurrent and deep networks. The device has 256 neurons and 128k analog synapses, and a prototype consumes approximately 4 mW with an area of 51.4 mm^2.

5.4 NeuroGrid

NeuroGrid is a neuromorphic system which was developed at Stanford to simulate massive biological neural network models in real time [28]. The system can simulate up to a million neurons with billions of synaptic connections, using 16 cores of 256 × 256 neurons each, embedded on a circuit board that consumes 3 W. It uses both analog and digital computation to mimic ion channel activity and structured connectivity patterns, respectively. NeuroGrid also uses shared circuits as axons, synapses, and dendrites to reduce transistor count. A software stack was developed including a user interface, hardware abstraction layer, and drivers. Using the interface, a user can configure neural networks, interact with simulations, and obtain real-time visualizations of system operation.

5.5 SpiNNaker

SpiNNaker (Spiking Neural Network Architecture) is a computing engine with a million cores, meant to simulate the behavior of up to a billion neurons in real time [29]. The modeling of large systems of spiking neurons is computationally expensive regarding processing power and communication. SpiNNaker is a massively parallel computer system developed to provide a cost-effective and flexible simulator for neuroscience simulation. It deploys a cluster of ARM9 cores, using packet communication via a custom massively interconnected fabric. With up to 1,036,800 ARM9 cores, SpiNNaker uses 64kbytes of data tightly coupled memory (DTCM) and 32kbytes of instruction tightly coupled memory (ITCM) for each core. Packets are of size 40 bits or 72 bits each, which are transmitted using a custom concurrent routing framework. In SpiNNaker operation, the computing engine consumes up to 90 kW of electrical power. The machine is built to mimic the brain’s biological structure and behavior, which exhibits massive parallelism and resilience to the failure of individual components. Given over one million cores, and one thousand simulated neurons per core, the SpiNNaker machine is capable of simulating one billion neurons, equivalent to over 1% of the human brain’s 85 billion neurons.

6 Conclusion

In this chapter, we have presented an overview of spiking neuron models, as well as some available SNN software and hardware simulation platforms. Also, we provided a detailed description of the STDP learning algorithm with application in handwritten digit recognition using Diehl and Cook experiment, which was implemented and tested with the well-known MNIST dataset [15]. In the above discussion, we have emphasized that SNNs are much more biologically realistic than ANNs, which is why they are used by neuroscientists in brain research. However, engineers have also shown an interest in SNNs because of their low power consumption and cognitive abilities. Recently, SNNs have found practical applications in autonomous robots and robotic limbs for quadriplegics [25, 26], and SNNs continue to be a popular area of research within the domain of robotics.

References

1. U. Güçlü, M. van Gerven, Probing human brain function with artificial neural networks, in *Comput. Model. Brain Behav.*, ed. by A.A. Moustafa, (2017), pp. 413–423. https://doi.org/10.1002/9781119159193.ch30
2. D. Soni, Spiking neural networks, the next generation of machine learning, *Data Science and Machine Learning* (2010). https://towardsdatascience.com/spiking-neural-networks-the-next-generation-of-machine-learning-84e167f4eb2b. Accessed 13 Oct 2019
3. H. Hazan et al., BindsNET: A machine learning-oriented spiking neural networks library in python. *Front. Neuroinform.* **12**, 1–18 (2018)
4. T. Mondeel, Modelling Neuronal Excitation: The Hodgkin-Huxley Model (2012)
5. J.B. Baladron, D.F. Javier, O. Faugeras, J. Touboul, Mean-field description and propagation of chaos in networks of Hodgkin-Huxley and FitzHugh-Nagumo neurons. *J. Math. Neurosci.* **2**(1), 1–67 (2012). https://doi.org/10.1186/2190-8567-2-10
6. J.L. Hindmarsh, R.M. Rose, A model of neuronal bursting using three coupled first order differential equations. Proc. R. Soc. Lond. B Biol. Sci. **221**(1222), 87–102 (1984)
7. L.F. Abbott, Lapicque's introduction of the integrate-and-fire model neuron (1907). Brain Res. Bull. **50**(5–6), 303–304 (1999)
8. P. Diehl, M. Cook, Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Front. Comput. Neurosci. **9**, 99 (2015)
9. Ş. Mihalaş, E. Niebur, A generalized linear integrate-and-fire neural model produces diverse spiking behaviors. Neural Comput. **21**(3), 704–718 (2009)
10. R.D. Vilela, B. Lindner, Comparative study of different integrate-and-fire neurons: Spontaneous activity, dynamical response, and stimulus-induced correlation. Phys. Rev. E Stat. Nonlinear Soft Matter Phys. **80**(3), 1–12 (2009)
11. A. Tavanaei, A. Maida, BP-STDP: Approximating backpropagation using spike timing dependent plasticity. Neurocomputing **330**, 39–47 (2019)
12. H. Markram, W. Gerstner, P.J. Sjöström, Spike-timing-dependent plasticity: A comprehensive overview. Front. Synaptic Neurosci., 2–5 (2012)
13. G. Bi, M. Poo, Synaptic modification by correlated activity: Hebb's postulate revisited. Annu. Rev. Neurosci. **24**(1), 139–166 (2001)
14. M. Stimberg, Brian 2 documentation—Brian 2 2.2.2.1 documentation (2016)
15. Y. LeCun, C. Cortes, C. Burges, The MNIST database of handwritten digits. Courant Inst. Math. Sci., 1–10 (1998)
16. D. Goodman, Brian: A simulator for spiking neural networks in Python. Front. Neuroinform. **2** (2008)
17. R. Brette, D.F.M. Goodman, Vectorized algorithms for spiking neural network simulation. Neural Comput. **23**(6), 1503–1535 (2011)
18. Neuron, Welcome to the community of NEURON users and developers (2018). http://www.neuron.yale.edu/neuron/. Accessed 26 Sept 2019
19. Genesis, GENESIS Resources (2019). http://genesis-sim.org/. Accessed 26 Sept 2019
20. M.-O. Gewaltig, M. Diesmann, NEST (Neural simulation tool). Scholarpedia **2**(4), 1430 (2007)
21. M. Smith, Self aware patterns, *SelfAwarePatterns* (2018). https://selfawarepatterns.com/2019/05/08/brain-inspired-hardware/. Accessed 30 Sept 2019
22. D. S. Modha, Introducing a brain-inspired computer: TrueNorth's neurons to revolutionize system architecture, *IBM Research* (2015). http://www.research.ibm.com/articles/brain-chip.shtml. Accessed 30 Sept 2019
23. A. S. Cassidy et al., Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores, in *Proceedings of the International Joint Conference on Neural Networks* (2013)
24. F. Akopyan et al., TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. IEEE Trans. Comput. Des. Integr. Circuits Syst. **34**(10), 1537–1557 (2015)

25. M. Feldman, IBM finds killer app for truenorth neuromorphic chip, *TOP500 Supercomputer Sites* (2016). https://www.top500.org/news/ibm-finds-killer-app-for-truenorth-neuromorphic-chip/. Accessed 30 Sept 2019
26. T. Hwu, J. Isbell, N. Oros, and J. Krichmar, A self-driving robot using deep convolutional neural networks on neuromorphic hardware, in *Proceedings of the International Joint Conference on Neural Networks*, vol 2017 (2017), pp. 635–641
27. N. Qiao et al., A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Front. Neurosci.* **9** (2015). https://doi.org/10.3389/fnins.2015.00141
28. B.V. Benjamin et al., Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. Proc. IEEE **102**(5), 699–716 (2014)
29. E. Painkras et al., SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation. IEEE J. Solid State Circuits **48**(8), 1943–1953 (2013)

An Efficient Algorithm for Mining Frequent Itemsets and Association Rules

Sallam Fageeri, Rohiza Ahmad, and Hitham Alhussian

1 Introduction

Association rule mining (ARM) is an increasingly popular approach in data mining. This popularity is motivated by the fact that traditional statistical techniques, data management tools, and decision support systems are unable to handle enormous amounts of data. The key to effective application of association rules is finding a representation of database items that enable rapid identification and reduced memory. In this chapter we make use a binary representation of data, which makes it possible to employ very fast bitwise operations to speed up processing. We verify this approach on several (publicly available) benchmark datasets, and show that this binary-based approach outperforms other algorithms in terms of reduced execution time and memory usage.

Association rule mining (ARM) on databases was first introduced by Agarwal et al. [1]. A recorded database transaction typically consists of a set of items (itemset), as shown in Table 1. We may imagine that each transaction corresponds to one

S. Fageeri (✉)
Department of Information Systems, CEMIS College, University of Nizwa, Nizwa, Sultanate of Oman
e-mail: sallam@unizwa.edu.om

R. Ahmad
Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, Bandar Seri Iskandar, Malaysia
e-mail: rohiza_ahmad@petronas.com.my

H. Alhussian
High Performance Cloud Computing Center (HPC3), Institute of Autonomous Systems, Universiti Teknologi PETRONAS, Bandar Seri Iskandar, Malaysia
e-mail: Seddig.alhussian@petronas.com.my

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_10

Table 1 Example of transaction records

TID	Itemsets			
100	A	B	C	
200	A	C	D	E
300	B	C	D	F
400	A	B	C	D
500	A	B	C	F

shopper's visit to a store, and each letter represents an item purchased by the shopper (A = apples, B = bananas, etc.)

ARM discovers frequent relationships between itemsets: for example, an association rule could be that transactions that contain $\{B, C\}$ also tend to contain D. Such rules may be useful in making profitable decisions in cross-marketing, promotional pricing, catalog design, customer segmentation, store layout, etc.

Formally, the association rule problem may be characterized as follows. Let $I = \{i_1, \quad i_2, \quad ..., \quad i_d\}$ be an itemset, and let D be a database consisting of N transactions denoted as $\{T_j, n = 1, \quad \ldots, \quad N\}$ such that $T_n \subseteq I \; \forall \; n$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X, Y \subseteq I$, and $X \cap Y = \varnothing$. For example, with reference to Table 1, we find that the rule $\{A\} \Rightarrow \{B\}$ holds true for transactions 100, 400, and 500, the rule fails for transaction 200, and the rule does not apply to transaction 300 (since item A is not in this transaction). Another example is $\{B\} \Rightarrow \{C, D\}$, which is true for transactions 300, 400, false for 100, 500, and does not apply to 200.

Association rules have two important attributes: *support* and *confidence*. Support indicates the proportion of *all* transactions in which the rule makes a correct prediction, while confidence is the proportion of *applicable* transactions in which the rule is correct. In the above examples, $\{A\} \Rightarrow \{B\}$ has support 3/5 (since 3 out of 5 total transactions represent a correct prediction) and confidence ¾ (since there are 3 correct predictions out of 4 applicable transactions). On the other hand, for similar reasons $\{B\} \Rightarrow \{C, D\}$ has support 2/5 and confidence 2/4.

To define these attributes mathematically, we must first define the *frequency of an itemset X in database D* as the proportion of transactions in D that contain X:

$$Freq\ (X, D) = \frac{\mid \{XT\} \mid}{\mid D \mid} \tag{1}$$

At this point we make note of an important property of frequency known as the *reverse monotonicity property:* if itemset Y is contained in itemset X, then the support of Y is greater than the support of X. Mathematically we may write this as:

$$Y \subseteq X \Rightarrow \quad Freq\ (Y, D) \geq Freq\,(X, D) \tag{2}$$

Given the definition of itemset frequency, we may define the *support* of a rule $X \Rightarrow Y$ *with respect to D* as the frequency of $X \bigcup Y$ in D:

$$Support\ (X \Rightarrow Y, D) = Freq\left(X \bigcup Y, D\right) \tag{3}$$

On the other hand, the *confidence* of a rule with respect to D is the proportion of database transactions that contain X which also contain $X \bigcup Y$:

$$Confidence\,(X \Rightarrow Y, D) = \frac{Freq\left(X \bigcup Y, D\right)}{Freq\,(X, D)} \tag{4}$$

To further illustrate these definitions, consider a database containing 100,000 database transactions, of which 2000 transactions contain *A* and *B* and 800 contain *A, B, C*. Then the association rule {*A, B*} => {*C*} has a support of 0.008 (800/100,000) and a confidence of 0.4 (800/2000).

If the support of a rule is very small, then it gives a correct conclusion in a very small proportion of transactions, and is thus not very useful. On the other hand, if the confidence is small, then it is not an accurate predictor. Thus, the search for association rules is typically limited to those rules with support greater than a given minimum value (*minsup*) and minimum confidence (*minconf*). The complexity of the ARM algorithm decreases with increasing *minsup* and *minconf*, because such increases reduce the search space. On the other hand, if *minsup* and/or *minconf* is set too high, then only the most obvious associations may be discovered and many interesting associations may be missed.

1.1 Problem Decomposition

Based on the description given in the previous section, we may divide the ARM problem into two phases:

Find all itemsets with frequency greater than or equal to *minsup.* These itemsets are referred to as "frequent itemsets."

For each itemset Z found in (1), find all association rules with support Z having confidence greater than or equal to *minconf.*

Phase (I) is the most computationally expensive step [2–4]. An exhaustive search for frequent itemsets (and their frequencies) requires comparing all transactions in the dataset with all possible itemsets, a total of $N \times 2^d$ comparisons. The number of comparisons grows exponentially with the number of items: in the data mining literature this is referred to as "the curse of dimensionality." Various strategies may be used to cut down the number of comparisons (see Sect. 3).

It follows that once the frequent itemsets $\{Z_1, \ldots, Z_J\}$ and their frequencies are found, phase (II) may readily be accomplished by computing the ratios freq(Z_j)/*freq*(Z_k) for each pair of frequent itemsets Z_j, Z_k with $Z_k \subset Z_j$. If the ratio is greater than *minconf*, then the association rule $Zk \Rightarrow Z_j \backslash Z_k$ (where "\" denotes the set difference operation) meets the minimum support and minimum confidence criteria. Therefore phase (II) involves computing at most $\sum_{\{j=0\}}^{J} 2^{|Z_j|}$ ratios: this number is typically much smaller than the number of comparisons required in phase I. It follows that most of the efforts in improving ARM are focused on improving the search for frequent itemsets.

2 Outline of the Binary-Based ARM Algorithm

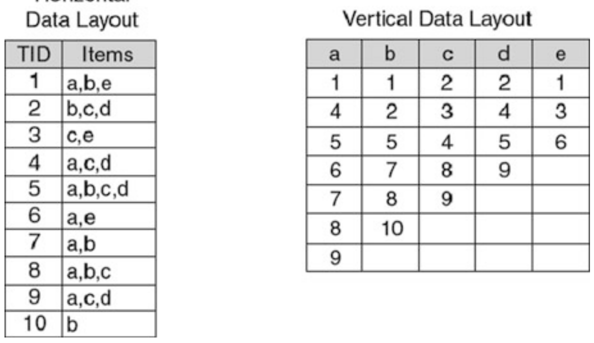
As mentioned previously, frequent itemset generation is the most resource-intensive part of association rules mining. In this section, we will describe an algorithm for frequent itemset generation with reduced complexity. The algorithm uses the following three strategies in order to reduce the complexity:

- Binary data representation (reduces the cost of scanning transactions)
- Bitwise operations (reduces the cost of comparisons)
- Pruning (reduces the number of candidate itemsets)

In the coming subsections we briefly describe these strategies, and in the next section we give a more detailed description of the algorithm.

2.1 *Binary Data Representation*

Transaction databases may be represented in many ways [5, 6]. The data representation can affect the process of computing the frequency of candidate itemsets, and also affect the input and output cost. Figure 1 shows two different approaches to representation of transactions databases. The horizontal layout lists the items associated with each transaction identifier (TID), and the vertical layout (also known as a "TID-list") lists the TID's associated with each item. The horizontal layout is used by the well-known Apriori algorithm, while TID-lists enable the computation



Horizontal Data Layout

TID	Items
1	a,b,e
2	b,c,d
3	c,e
4	a,c,d
5	a,b,c,d
6	a,e
7	a,b
8	a,b,c
9	a,c,d
10	b

Vertical Data Layout

a	b	c	d	e
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

Fig. 1 Horizontal and vertical data format

Table 2 Binary representation of database *BitSetDB*

TID (j)	BitSetDB (j)
0	11001
1	01110
2	00101
3	10110
4	11010
5	10001
6	11000
7	11100
8	10110
9	01000

Table 3 Array of frequencies *Freq* derived from the binary database *BitSetDB*

0	1	2	3	4
7	6	5	4	3

of itemset frequencies by intersecting the TID's listed for each item in the itemset. TID-lists tend to be memory intensive (since each entry is a TID, which in general is larger than an item ID), so in some cases refining techniques are required to compress TID-lists.

A third alternative representation is the binary representation shown in Table 2 as the array *BitSetDB*. This resembles the horizontal format in that rows correspond to transactions; but instead of listing the items involved in each transaction, a binary string is used to indicate inclusion of items within each transaction's itemset as follows. Each item in I is assigned an index from 0 to $d - 1$, where $d = |I|$. For each TID, a d-bit string is listed: if the item indexed by j is included within the transaction, then the j'th bit is 1; otherwise the j'th bit is 0. For example, in Table 1 the first row gives *BitSetDB*(0) = '11001', meaning that transaction 0 contains items indexed by 0,1, and 4. Similarly, *BitSetDB*(3) = '10110', which implies that items 0, 2, 3 are included in transaction 3.

By counting the frequency of 1's at each bit position, an array *Freq* may be constructed that records frequency of all items in the database (see Table 3). Note that the leftmost bit corresponds to bit position 0.

The mathematical formula for the entries of *Freq* is

$$Freq(j) = \sum\nolimits_{i=1}^{N} BitSetDB(i)(j), \quad j = 1, 2, \ldots, m \tag{5}$$

2.2 *Masks and Bitwise Operations*

Given the binary data representation described above, efficient search for frequent itemsets is enabled through the use of *masks*. An example of a mask is shown in

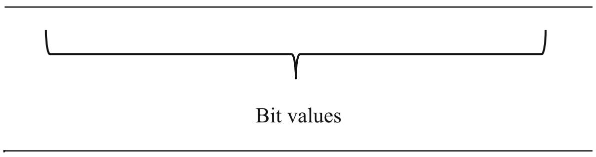


Fig. 2 A *bitSetMask* for identifying an itemset with 2 items

Fig. 2. The mask length corresponds to the parameter m, which is the total number of items included in the dataset (and is also the length of the binary representation of each transaction). In Fig. 2, bits 0 and 2 are set to True and all others to False. When this mask is applied to a binary transaction record, the result will be equal to the mask if and only if items 0 and 2 are in the transaction.

In general, given an itemset $I = \{i_1, i_2, \ldots, i_d\}$, a mask may be constructed for detecting that itemset according to the following specification. Letting b_j denote the j'th bit of the mask, we let:

$$b_j = \begin{cases} 1 \text{ if } \; j = i_k \; \text{ for some } \; k, 1 \le k \le d \\ 0 \text{ otherwise} \end{cases}$$

When this mask is ANDed with a binary transaction, then the result is equal to the mask if and only if the transaction contains the itemset I. It follows that in order to identify the support of any itemset, all that is needed is to AND the itemset's mask with all transactions, and note when the result is equal to the itemset's mask. This fact may be used to rapidly identify transactions that contain itemsets. It remains to specify a systematic order in which different itemsets are tested so as to identify all large frequent itemsets as quickly as possible. This procedure is described in detail in the next section.

2.3 Itemset Pruning via Merging Operation

The algorithm searches for itemsets by size: first frequent items, then frequent 2-itemsets (i.e., itemsets with 2 items), then frequent 3-itemsets, and so on. As the algorithm tests larger and larger itemsets, the number of possible itemsets increases exponentially with the number of items. For example, in the case where there are 10 total items, then there are 10 1-itemsets, (10)(9)/2 = 45 2-itemsets, (10)(9)(8)/(3)(2)(1) = 120 possible 3-itemsets, and so on. In order to reduce the number of k-itemsets tested, we may make use of the results obtained from $(k-1)$-itemsets using a "merging" technique (this technique was first used in the Apriori algorithm designed by Agrawal et. al. [6]).

We may illustrate this technique with an example. Suppose $\{A, B, C\}$, $\{A, B, D\}$, $\{A, C, D\}$, $\{A, C, E\}$, $\{A, D, E\}$, $\{B, C, E\}$, $\{B, D, E\}$, and $\{C, D, E\}$ have been identified as the frequent 3-itemsets for a particular transaction database. For any 4-itemset to be frequent, all 3-item subsets must also be frequent. For example, for

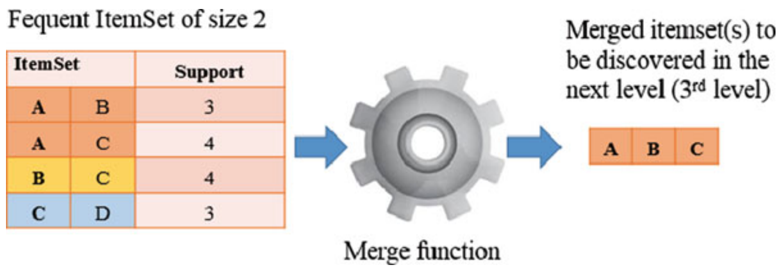

Fequent ItemSet of size 2
ItemSet
Support
A B 3
A C 4
B C 4
C D 3
Merge function
Merged itemset(s) to be discovered in the next level (3rd level)
A B C

Fig. 3 Merging frequent 2-itemsets to find candidate 3-itemsets

{*A, B, C, D*} to be frequent, then {*A, B, C*}, {*A, B, D*}, {*A, C, D*}, and {*B, C, D*} must also be frequent. But since {*B, C, D*} is not frequent, this possibility is eliminated. Similarly, for {*B, C, D, E*} to be frequent, then {*B, C, D*}, {*B, C, E*}, {*B, D, E*}, and {*C, D, E*} must all be frequent. Once again, this possibility is eliminated because {*B, C, D*} is not frequent. In this example, only {*A, C, D, E*} is a possible 4-itemset because {*A, C, D*}, {*A, C, E*}, {*A, D, E*}, and {*C, D, E*} are all frequent. So an easy test for whether a 4-itemset may be frequent is to check that all 3-itemsets obtained by removing a single item are also frequent. (This test does not *guarantee* that the 4-itemset is frequent, but it does disqualify many candidates that will thus not need to be checked.)

This rule may be readily generalized to n-itemsets for any value of k with $k > 2$. A necessary condition for $\{I_1, I_2, \ldots, I_k\}$ to be frequent is that all k itemsets obtained by removing a single I_j from the set must still be frequent.

A rule that is easier to implement (but less thorough) is to identify pairs of $n - 1$ itemsets that have the same $n - 2$ initial items, and then merge all such pairs to form the list of candidate n-itemsets. In the above example, the 3-itemsets {*A, B, C*} and {*A, B, D*} have {*A, B*}, in common so that {*A, B, C, D*} is a candidate 4-itemset. Also, {*A, C, D*} and {*A, C, E*} have {*A, C*} in common so that {*A, C, D, E*} is a candidate 4-itemset.

A simpler example of merging 2-itemsets to form candidate 3-itemsets is shown in Fig. 3.

2.4 *Binary-Based Algorithm Description*

The rest of the binary-based algorithm can be described through the following steps.

2.4.1 Top-Level Description

Figure 4 gives a flowchart for the proposed algorithm, which consists of two phases, as indicated in Sect. 2.1. Phase I involves finding itemsets whose support

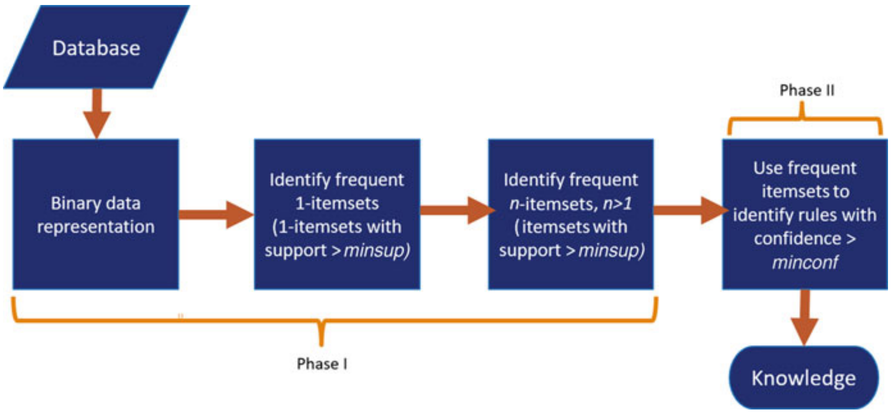

Database
Binary data representation
Identify frequent 1-itemsets (1-itemsets with support > minsup)
Identify frequent n-itemsets, n>1 (itemsets with support > minsup)
Use frequent itemsets to identify rules with confidence > minconf
Phase II
Phase I
Knowledge

Fig. 4 Top-level flowchart of the approach

Table 4 Sample transactions

TID	Itemset			
100	*A*	*B*	*C*	
200	*A*	*C*	*D*	*E*
300	*B*	*C*	*D*	*F*
400	*A*	*B*	*C*	*D*
500	*A*	*B*	*C*	*F*

is greater than the user-specified minimum *support* (such itemsets are called "large frequent itemsets"). This phase (which is the most computationally intensive part of the calculation) divides naturally into three steps, as shown in the figure. Phase II extracts rules which meet the support criterion and also possess minimum *confidence* greater than the user-specified minimum *confidence*.

The four procedures shown in Fig. 5 may be described as follows:

- *Binary data representation*: obtains the list of items in the database, and generates the binary representation of the database transactions.
- *Identify frequent 1-itemsets*: scans the database, counts the frequency of all items in the itemset using Eq. (3), and creates a list of items whose support is greater than or equal to *minsup*.
- *Identify frequent n-itemsets*: generates the lists of all frequent itemsets with support greater than or equal to *minsup*.
- *Identify frequent itemsets with minimum confidence*: generates all the association rules based on the frequent itemsets that have been generated in the previous procedures.

In the following discussion, the above procedures will be illustrated using the database of sample transactions shown in Table 4.

2.4.2 Binary Data Representation

This procedure reads transactions from the stored database and generates a database containing the binary representation of the transactions, as described in Sect. 3. Figure 5 illustrates this procedure in a flowchart form (Table 5).

2.4.3 Procedure for Finding Frequent 1-Itemsets

The binary representation has the advantage that the supports of all items may be found by independently examining the contents of items' columns, as shown in Fig. 6. As a result, supports of multiple items may be computed in parallel.

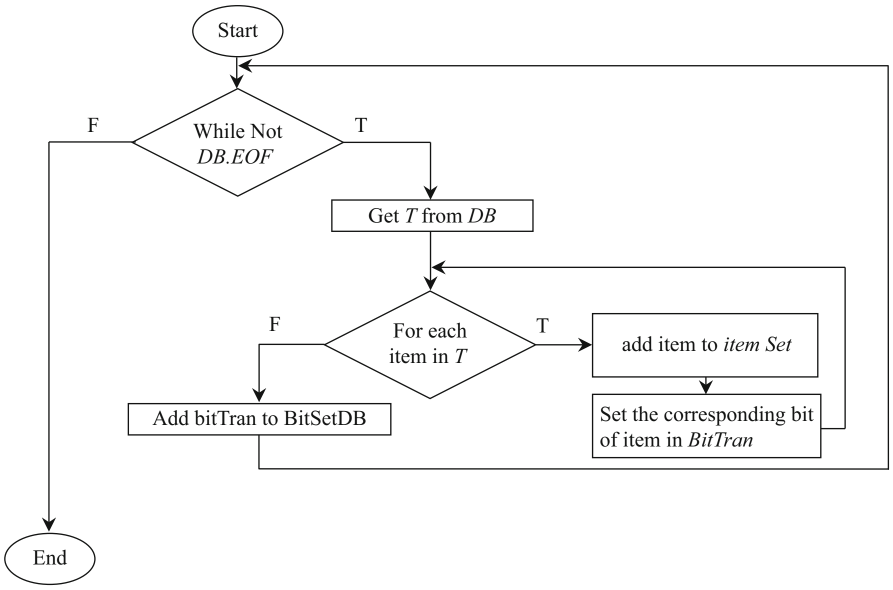
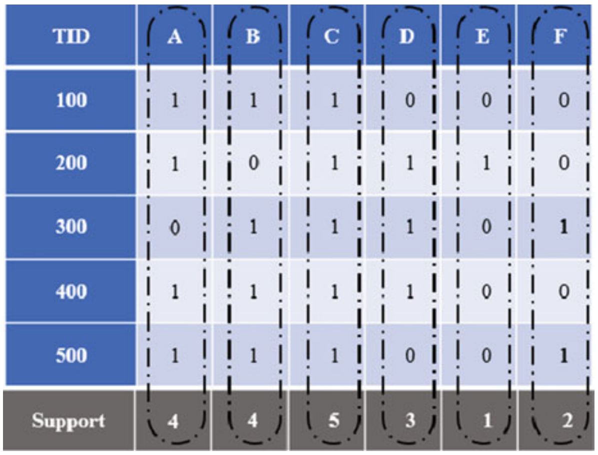


Fig. 5 Flowchart for creating binary database (*DB* = database, *T* = transaction, *BitTran* = binary transaction, *BitSetDB* = binary database)

Table 5 Shows the output of this procedure applied to the original database in Table 1

TID	*A*	*B*	*C*	*D*	*E*	*F*
100	1	1	1	0	0	0
200	1	0	1	1	1	0
300	0	1	1	1	0	1
400	1	1	1	1	0	0
500	1	1	1	0	0	1



TID	A	B	C	D	E	F
100	1	1	1	0	0	0
200	1	0	1	1	1	0
300	0	1	1	1	0	1
400	1	1	1	1	0	0
500	1	1	1	0	0	1
Support	4	4	5	3	1	2

Fig. 6 Support counting column by column using the binary representation

Table 6 Sample output from the procedure for finding frequent 1-itemsets, with *minsupp* set equal to 0.5

	Items	D	A	F	C	B	E
	Freq Support	3 0.6	4 0.8	2 0.4	5 1.0	4 0.8	1 0.2
	Frequent items	D	A		C	B	

Grayed item are identified as infrequent

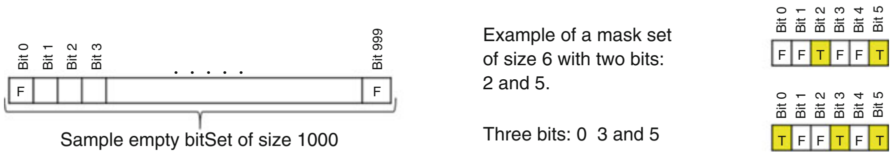

Bit 0
Bit 1
Bit 2
Bit 3
. . . . .
Bit 999
F
F
Sample empty bitSet of size 1000
Example of a mask set of size 6 with two bits: 2 and 5.
Bit 0 Bit 1 Bit 2 Bit 3 Bit 4 Bit 5
F F T F F T
Three bits: 0 3 and 5
Bit 0 Bit 1 Bit 2 Bit 3 Bit 4 Bit 5
T F F T F T

Fig. 7 Examples of masks: (*left*) unset mask; (*right*) masks set for itemsets of 2 and 3 bits

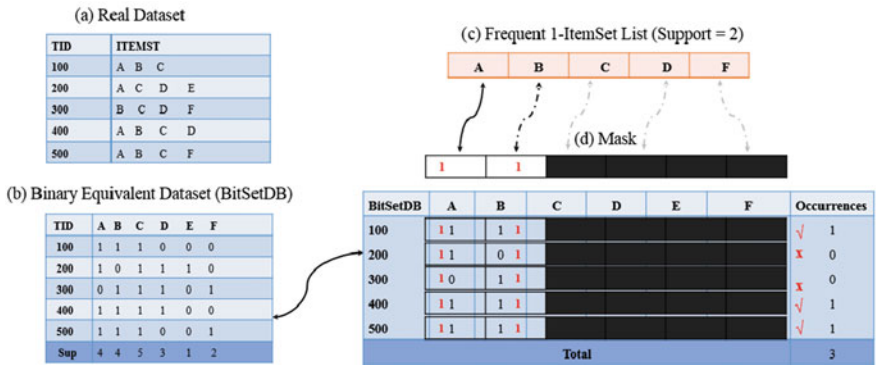
After counting the 1's in each column, the support is computed by dividing by the total number of transactions as indicated in Eq. (3). The result of these operations on the binary database in Table 2 is shown in Table 6 (Fig. 7).

2.4.4 Procedure for Generating Frequent Itemsets with Multiple Items

The identification of frequent itemsets is done in two stages. The first stage calculates the frequent itemsets of size 2 (which are designated as 2-itemsets), while the second stage calculates the frequent itemsets of size >2. As indicated in Sect. 2.8, the current implementation uses masks to calculate itemset frequencies. A mask is a vector (linear array) of bits with size equal to the number of items (which is also equal to the width of the binary database). Figure 8 shows an example of a mask in the case where the transactions contain 1000 items. To calculate the support of any itemset, the bits corresponding to the itemset are set as “T,” while all other bits are “F.”

Once the mask bits are set for a particular itemset, the support of the itemset is counted by comparing the mask with each transaction in the binary database and performing logical operations. If the bitwise AND of the mask with the transaction agrees with the transaction, then +1 is added to the support count, as shown in Fig. 8c, d.

Figure 9 gives a flowchart showing the process for determining 2-itemsets. Two nested loops are used to generate all possible pairs of frequent items (*i, j*); for each of these pairs, the mask with nonzero bits at positions *i* and *j* is created; then the mask is ANDed with each transaction in the database, and transactions where the AND result agrees with the mask are counted as containing the itemset (*i, j*) (Table 7).


(a) Real Dataset
(b) Binary Equivalent Dataset (BitSetDB)
(c) Frequent 1-ItemSet List (Support = 2)
(d) Mask

(a) Real Dataset

TID	ITEMST
100	A B C
200	A C D E
300	B C D F
400	A B C D
500	A B C F

(b) Binary Equivalent Dataset (BitSetDB)

TID	A	B	C	D	E	F
100	1	1	1	0	0	0
200	1	0	1	1	1	0
300	0	1	1	1	0	1
400	1	1	1	1	0	0
500	1	1	1	0	0	1
Sup	4	4	5	3	1	2

(c) Frequent 1-ItemSet List (Support = 2)

A	B	C	D	F

(d) Mask

1	1				

BitSetDB	A	B	C	D	E	F	Occurrences
100	1 1	1 1					√ 1
200	1 1	0 1					x 0
300	1 0	1 1					x 0
400	1 1	1 1					√ 1
500	1 1	1 1					√ 1
Total							3

Fig. 8 ANDing the mask for the 2-itemset (A,B) with all transactions in the binary database: (**a**) original dataset; (**b**) dataset converted to binary format; (**c**) list of frequent itemsets; (**d**) binary mask for (A,B), applied to all transactions in the database (with total count of occurrences)

Table 7 Shows the result of mask generation for the 2-itemset (A, D), and subsequent ANDing with transactions using the sample binary database shown in Table 2

BitSet	A	B	C	D	E	F	Support
Mask	1			1			2

BitSetDB	A	B	C	D	E	F	Occurrences
100	1	1	1	0	0	0	x
200	1	0	1	1	1	0	√
300	0	1	1	1	0	1	x
400	1	1	1	1	0	0	√
500	1	1	1	0	0	1	X

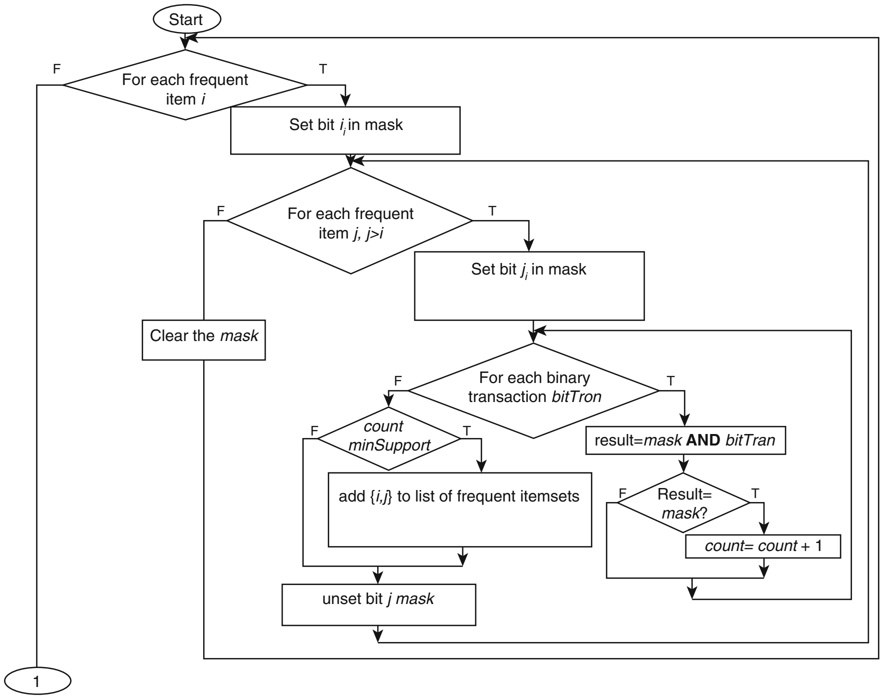

Start
For each frequent item *i*
F
T
Set bit i_i in mask
For each frequent item *j, j>i*
Set bit j_i in mask
Clear the *mask*
For each binary transaction *bitTron*
result=*mask* **AND** *bitTran*
Result= *mask*?
count= *count* + 1
count *minSupport*
add {*i,j*} to list of frequent itemsets
unset bit *j mask*
1

Fig. 9 Flowchart showing generation of frequent 2-itemsets

In the second stage, lists of frequent itemsets with more than 2 items are generated. A *while* loop is used to search for frequent n-itemsets, conditioned on whether the set of frequent $n - 1$ itemsets is non-empty. If so, then this set is ordered in lexicographic order. Another pair of nested loops scans through the ordered $n - 1$ itemsets to identify pairs that can be merged, as described in Sect. 2.8. Based on these pairs, a list of candidate n-itemsets is generated. Once again, masks are used to count the frequencies of these candidate itemsets.

2.4.5 Phase II: Extracting Association Rules

Based on the frequent itemsets discovered in phase I, in phase II the association rules are extracted [1]. We first show how this is done in a specific example. Supposing that $\{A, B, C\}$ is a frequent itemset, then all possible rules involving the 3 items A, B, C are shown in the tree diagram of Figure 10. The confidences for all of these rules may be found by taking ratios of frequencies that have already been computed. For example, the confidence of the rule $\{A, B\} \Rightarrow C$ is $Freq(\{A, B, C\})/Freq(\{A, B\})$.

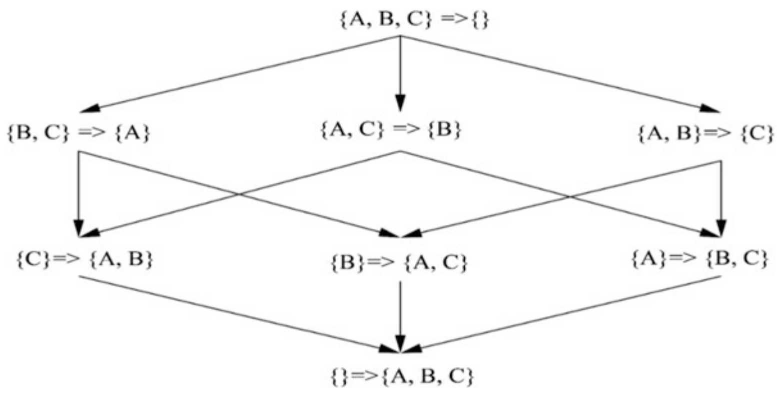


Fig. 10 Example of extracting association rules

Table 8 Synthetic datasets characteristics

Dataset	#Items	#Transactions	Average # of transactions	Max $\|T\|$	Type
T10I4D100K	1000	100,000	10	29	Sparse
T40I10D100K	1000	100,000	40	77	Sparse

Table 9 Real datasets characteristics

Dataset	#Items	#Transactions	Average items/transact.	Max $\|T\|$	Type
Retail	16470	88,162	10	76	Sparse
Accidents	468	340,183	33	51	Dense
Chess	76	3196	37	37	Dense
Connect	130	67,557	43	43	Dense
Pumsb	7117	49,046	74	74	Dense

2.5 *Datasets*

All datasets used in the evaluation of the algorithm are publicly available in [7]. Both synthetic and real datasets were included.

The synthetic datasets are provided by the QUEST generator of data generated from IBM’s Almaden research lab. The method used to generate the synthetic datasets is described in Agrawal et al. [8]. Table 8 shows the characteristics of the synthetic datasets.

The real datasets are provided by the frequent itemsets mining dataset repository, based on workshops conducted in [7]. Table 9 shows the characteristics of the real datasets.

The following characteristics of these dataset may be noted:

1. T40I10D100K datasets are sparse, but they produce a large number of frequent items, when the minimum support threshold value is small.

2. T10I4D100K, T20I6D100K, and Retail datasets are very sparse and have a small number of frequent itemsets even though the minimum support threshold value is small.
3. Accidents, Chess, and Connect are unstructured datasets containing large number of transactions and a small number of unique items.
4. Pumsb is an unstructured dataset containing a large number of transactions, as well as a large number of unique items.

2.6 Software and Hardware Specifications

All algorithms were implemented in Java language. The hardware used for the experiments is DELL PRECISION T1700 machine equipped with 4 cores running at 3.10 GHz speed and 8 GB of RAM memory. The operating system installed in the hardware is Windows 7 64 bits.

2.7 Execution Time Benchmarking

Figure 11 shows the execution times for five different frequent itemset algorithms for four different benchmark datasets. In each graph, execution time is shown as a function of support: as support decreases, the number of frequent itemsets increases, and the execution time increases correspondingly. The binary-based algorithm consistently has the lowest execution times—in some cases, less than half the execution time of its nearest competitor. This improvement can be attributed to the faster execution of bitwise operations used in the binary-based algorithm, compared to operations used in the other algorithms. As far as the other algorithms, H-mine has the highest times (usually more than 4 times slower than binary-based) typically followed by Apriori (2–3 times worse than binary-based). The relative efficiency of FP-growth and Eclat depends on the dataset, as different types of datasets make different demands on the construction of the data structures used by these algorithms.

2.8 Memory Usage Benchmarking

Figure 12 shows the memory usage for five different frequent itemset algorithms for four different benchmark datasets. As before, memory is shown as a function of support: as support decreases, the number of frequent itemsets increases, and the memory typically increases as well. The graphs show that the binary-based method consistently uses the smallest amount of memory. This reflects the efficiency of the

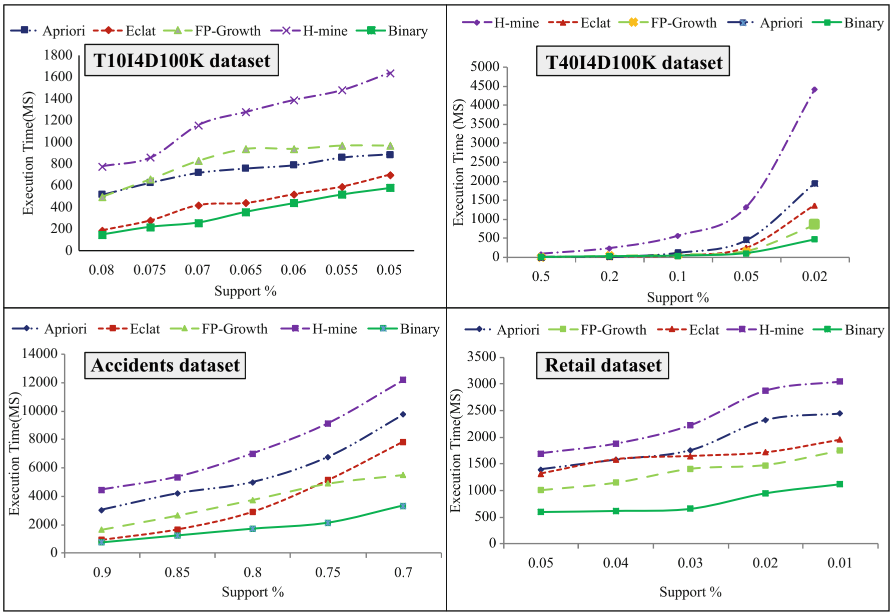


Fig. 11 Execution times for different frequent itemset algorithms for four different benchmarks: (*top left*) T10I4D100K synthetic dataset; (*top right*) T40I4D100K synthetic dataset; (*bottom left*) Accidents dataset; *(bottom right)* Retail dataset

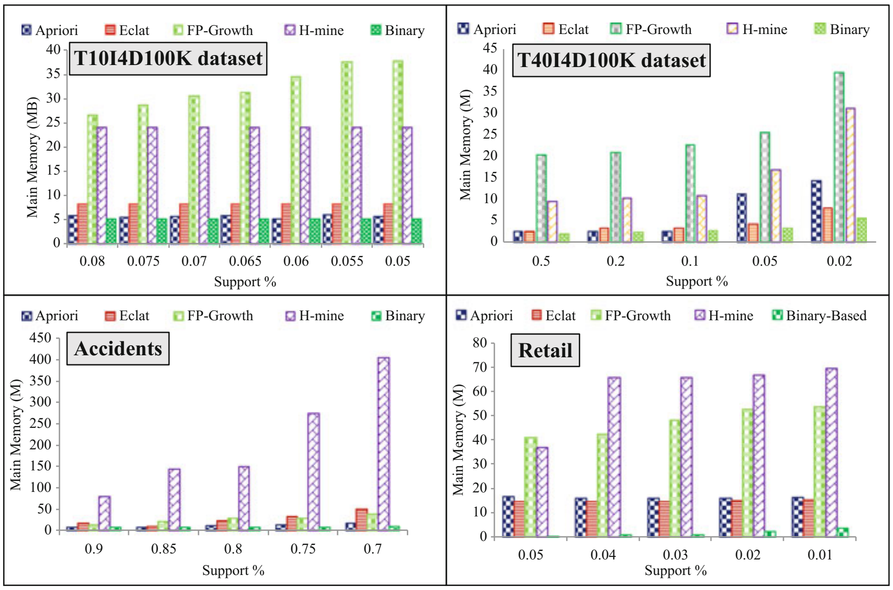


Fig. 12 Memory usage (in megabytes) for different frequent itemset algorithms for four different benchmarks: (*top left)* T10I4D100K synthetic dataset; (*top right*) T40I4D100K synthetic dataset; (*bottom left)* Accidents dataset; *(bottom right)* Retail dataset

bitset representation of the database. Unlike FP-growth and H-mine, the algorithm does not require the construction of accessory data structures (e.g., FP-trees). Thus, the memory used is fairly stable, even as the support (and hence the number of frequent itemsets) increases.

2.9 Summary

The experimental setup as well as the evaluated performance of the binary-based approach against the state-of-the-art algorithms is presented. The performance has been evaluated using two important factors: execution time and memory usage. Different datasets with different characteristics have been used to test the performance of the proposed binary-based approach. With regard to execution times, binary-based approach as a new technique employing binary representation of the dataset as well as bitwise operations to manipulate the data always achieves better performance in all datasets for all high and low support values. With regard to memory consumption, the binary-based approach also outperforms all other algorithms and achieved the lowest memory usage in almost all datasets. As previously mentioned, this is likely due to the efficiency of the binary format, which stores each transaction in the database as a fixed-width binary bitset, with individual items represented by 0s and 1s.

References

1. R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in *ACM SIGMOD Record*, (1993), pp. 207–216
2. Y. Le Bras, P. Lenca, S. Lallich, On optimal rule mining: a framework and a necessary and sufficient condition of antimonotonicity, in *Advances in Knowledge Discovery and Data Mining*, (Springer, 2009), pp. 705–712
3. N. Vanetik, Analyzing closed frequent itemsets with convex polytopes, http://arxiv.org/abs/1203.4380 (2012), pp. 1–14
4. S.K. AV, A. Al-Rabea, I.M. El Emary, Frequent itemsets mining: an efficient graphical approach. World Appl. Program. **1**, 330–338 (2011)
5. M.J. Zaki, Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. **12**, 372–390 (2000)
6. T. Pang-Ning, M. Steinbach, V. Kumar, Introduction to data mining, in *Library of Congress* (2006)
7. B. Goethals, Frequent itemset mining dataset repository. Available: http://fimi.cs.helsinki.fi/data/ (2013)
8. R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in *Proceedings of the 20th International Conference Very Large Data Bases, VLDB*, (1994), pp. 487–499

Receiver Operating Characteristic Curves in Binary Classification of Protein Secondary Structure Data

Saad Subair and Christopher Thron

1 Introduction

The most basic classification problem involves distinguishing between two classes of objects based on the value of a single measurement or calculated value. This so-called *dichotomous (or binary) classification* is a convenient and powerful tool for decision-making, although it may introduce distortions [1, 2].

Typically, classification involves setting a threshold, and for each object the choice of class is determined by whether the value found for that object is above or below the threshold. It follows that evaluating the effectiveness of a classifier and choosing the most appropriate threshold are key considerations in the practical use of binary classification. For this purpose, Receiver Operating Characteristics (ROC) curves are extremely useful for assessing the performance of classifiers. ROC curves are well known in biological and medical decision-making, and have been increasingly adopted as a tool for analysing and visualizing many aspects of machine learning algorithms or methods.

We will introduce the concepts involved in the construction and use of ROC curves by means of a specific example, namely, classification of protein shape.

S. Subair (✉)
College of Computer Studies, International University of Africa (IUA), Khartoum, Sudan

C. Thron
Department of Science and Mathematics, Texas A&M University-Central Texas, Killeen, TX, USA

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*,
Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_11

2 Classification of Protein Shape

The structure of any protein can be classified as either helical (H), strands (E), or coil (C). In biological systems, the proteins present in the system consist of about 30%, 20%, and 50% of *H*, *E*, and *C* respectively [3]. In this context, distinguishing any one of the three types from the other two is a binary classification problem. Note that rates of successful prediction are strongly dependent on the prevalence of the classes that are being distinguished. For example, a trivial prediction algorithm which assigns all proteins to *C* gives correct prediction rates of approximately 50%, even though no measurements are being performed. On the other hand, an equally trivial prediction algorithm that assigns all proteins to *H* will be correct only 30% of the time. The two methods are equally uninformative, although one appears to give a better result than the other.

Supposing that we have an amino acid sequence of length n, then we have n structures that can have values *H*, *E*, or *C*. We will consider the case of a binary classification between *C* and non-*C*. Accordingly we define $d_1, d_2, \ldots d_{n.}$ such that $d_j = 1$ if the jth structure is non-*C*, and $d_j = 0$ otherwise. We similarly define g_1, $g_2, \ldots g_n$ as the classifier's outputs for the n structures: $g_j = 1$ if the jth structure is classified as non-C, and $g_j = 0$ otherwise. In this situation, there are four possible outcomes for the jth amino acid in the sequence:

- $d_j = 1$, $g_j = 1$ (*true positive:* jth structure is correctly identified as *C*),
- $d_j = 0$, $g_j = 0$ (*true negative:* jth structure is correctly identified as non-*C*),
- $d_j = 0$, $g_j = 1$ (*false positive:* jth structure is incorrectly identified as *C*),
- $d_j = 1$, $g_j = 0$ (*false negative:* jth structure is incorrectly identified as non-*C*),

3 Sensitivity and Specificity

Based on the four possible outcomes we have just described, we define the following four numbers that completely characterize the classification result:

TP = the number true positives.
TN = the number of true negatives.
FP = the number of false positives.
FN = the number of false negatives.

From these definitions it is clear that every structure must give exactly one of the four results, so we have:

$$n = TP + TN + FP + FN.$$

The four numbers are often arranged into a 2×2 *contingency table* (or *confusion matrix)* as shown in Table 1.

Table 1 Contingency table for classification results

		Prediction (g_j)	
		C	non-C
Actual (d_j)	C	TP	FN
	Non-C	FP	TN

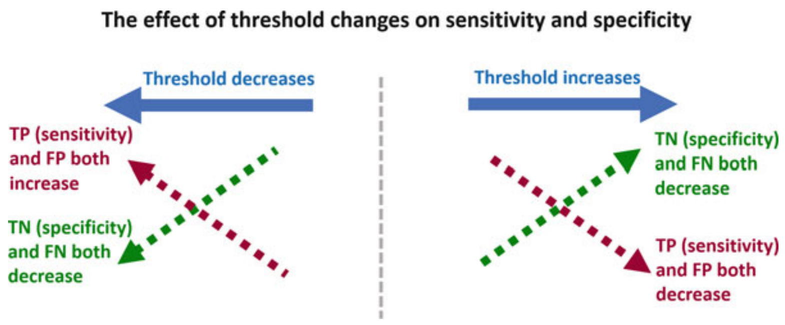

The effect of threshold changes on sensitivity and specificity
Threshold decreases
TP (sensitivity) and FP both increase
TN (specificity) and FN both decrease
Threshold increases
TN (specificity) and FN both decrease
TP (sensitivity) and FP both decrease

Fig. 1 Relation between threshold changes and *TP, FP, TN,* and *FN*

We may define the *sensitivity* as the proportion of C structures that are correctly identified as C. The sensitivity is obtained by dividing the true positives by the total number of positives:

$$\text{Sensitivity (also called hit rate or true positive rate)} = TP/(TP + FN)\,.$$

We may also define the *specificity* as the proportion of non-C structures that are correctly identified as non-C. The specificity is obtained by dividing the true negatives by the total number of negatives:

$$\text{Specificity (also called selectivity or true negative rate)} = TN/(FP + TN)\,.$$

In other situations where either the actual state d_j or the output g_j is not binary, then the situation is more complex and four numbers are not enough to summarize the situation. If g_j is not binary, binary predictions can still be obtained by using a cut-off threshold: the prediction is rendered as 1 if g_j is above a predetermined threshold value, and 0 otherwise (in this paper we will adopt the convention that if g_j is equal to the threshold, then the value is 0). The numbers *TP, TN, FP*, and *FN* will then vary with the threshold choice. As the threshold is decreased, more and more structures will exceed the threshold, and the number of true positives (*TP*) increases—but so does the number of false positives (*FP*). Conversely, as the threshold is increased, the number of true negatives (*TN*) increases, but so does the number of false negatives (*FN*). There is always trade-off between the proportions of false positives and false negatives produced by the algorithm or the classifier (see Fig. 1).

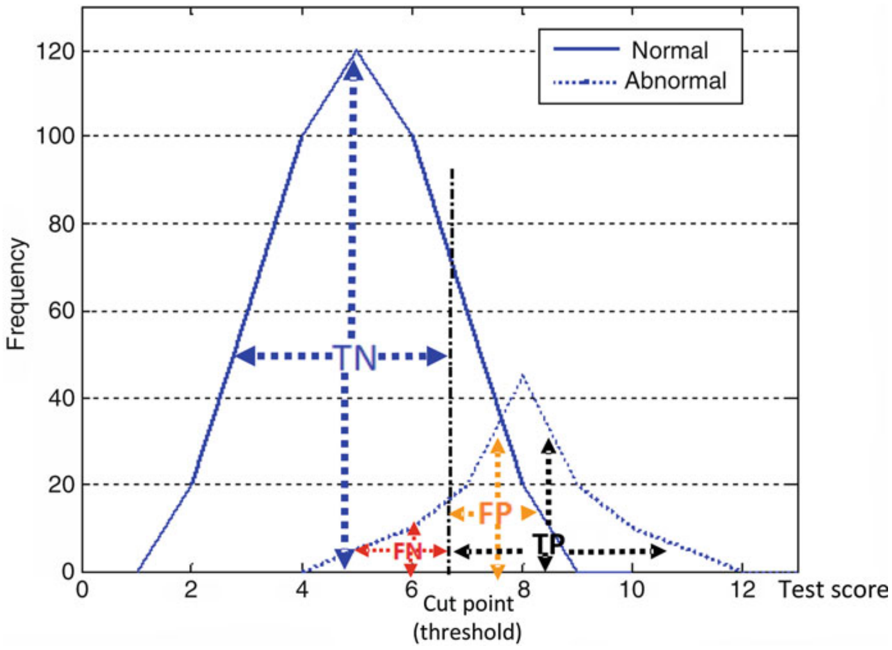
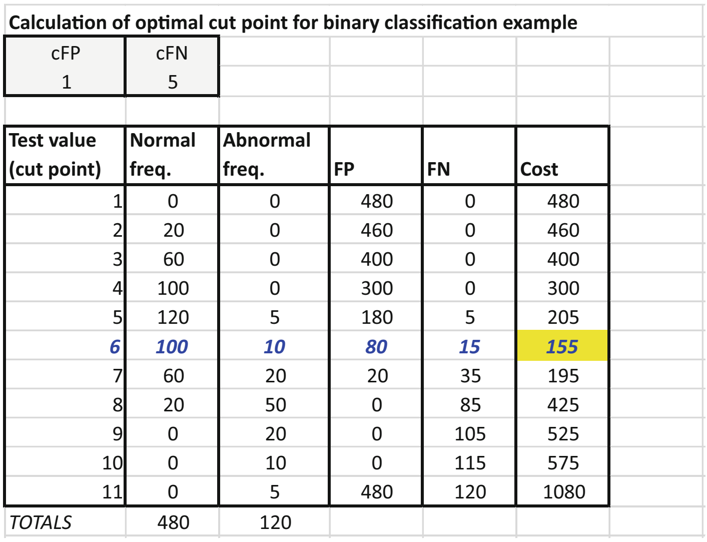


Fig. 2 Cut point for classification between normal and abnormal individuals

An alternative representation of the four numbers TP, FP, TN and FN is shown in Fig. 2. The two curves show the frequencies of normal and abnormal structures in a sample of about 600 structures, as a function of their scores on a certain test (for example, the figure shows that there were 100 normal and about 10 abnormal structures with a score of 6, while there were 20 normal and about 45 abnormal structures with a score of 8). The test score can be used to predict normal from abnormal by setting a threshold (or cut point), and classify individuals above the cut point as abnormal, and individuals below the cut point as normal. The position of the cut point will determine the number of true positive, true negatives, false positives, and false negatives as shown in Fig. 2. Consequently, the sensitivity and specificity of the test based on a particular threshold can also be estimated from *TP*, *FP*, *TN*, and *FN* using the formulas given above.

Additional information is required to identify the best threshold, namely the relative cost of *FP* and *FN* errors. Assigning values to these costs are complex and subjective and dependent upon the context within which the classification rule will be used. However, once the relative costs have been determined, the calculation of the best threshold is straightforward. Supposing that the costs of *FP* and *FN* errors are c_{FP} and c_{FN}, then the cost due to the errors for the give sample is:

$$\text{Error cost} = c_{FP} \cdot FP + c_{FN} \cdot FN.$$



Calculation of optimal cut point for binary classification example

cFP	cFN
1	5

Test value (cut point)	**Normal freq.**	**Abnormal freq.**	**FP**	**FN**	**Cost**
1	0	0	480	0	480
2	20	0	460	0	460
3	60	0	400	0	400
4	100	0	300	0	300
5	120	5	180	5	205
6	***100***	***10***	***80***	***15***	***155***
7	60	20	20	35	195
8	20	50	0	85	425
9	0	20	0	105	525
10	0	10	0	115	575
11	0	5	480	120	1080
TOTALS	480	120			

Fig. 3 Calculation of costs for different cut points, given specified values of c_{FP} and c_{FN}. The lowest cost is achieved when 6 is chosen as cut point. Since test values are integers, any non-integer between 6 and 7 will also give the same results

It is easy to find (using a computer if necessary) the cut point that yields the smallest error cost. For example, consider the sample shown in Fig. 2, and suppose that c_{FN} is determined to be 5 times larger than c_{FP}. Then the best cut point is between 6 (inclusive) and 7 (exclusive), as shown in the spreadsheet in Fig. 3. Although this cost calculation was only done for one sample size, all costs scale equally for different sample sizes, so the same cut points will be optimal regardless of sample size.

Alternatively, the optimum cut point can also be calculated on the basis of the following information:

- Sensitivity (denoted by *sens*).
- Specificity (denoted by *spec*).
- Prevalence of the normal case (given as a fraction of the total population size: denoted by p).
- Relative costs of *FP* and *FN* (denoted by c_{FP} and c_{FN}).

Based on this information, the optimum cut point is the one that minimizes the expected error cost per sample, given by:

$$\text{Expected error cost per sample} = (c_{FP})\,(1 - sens)\,(p) + (c_{FN})\,(1 - spec)\,(1 - p)$$

4 Receiver Operating Characteristics (ROC) Curves

The Receiver operating characteristics (ROC) curve summarizes the trade-off between sensitivity and specificity for different choices of threshold. The curve is constructed by plotting the false positive rate (equal to (1 − specificity)) on the horizontal axis, versus true positive rate (equal to sensitivity) on the vertical axis. As shown in Fig. 1, true and false positive rates have the same trend, and thus the ROC curve will always have a positive slope. Furthermore, the true positive rate will always be larger than the false positive rate, for otherwise, the test is worse than random guessing! It follows that the ROC curve will always lie above the 45 degree line. An example ROC curve, calculated from the data shown in Fig. 2 is shown in Fig. 4.

The closer the curve follows the left-hand border and then the top border of the chart, the more accurate the test, while the closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

Although the ROC curve in itself is not sufficient to determine an optimal cut point, it does accurately represent the possibilities of sensitivity and specificity that can be achieved. For example, if a sensitivity of 0.9 is desired, the curve in Fig. 4 shows that the achievable specificity is roughly 1–0.15, or 0.85.

The area under the curve (AUC) is a measure of the overall test accuracy. The AUC can be interpreted as the *average* sensitivity, where the average is assessed over all possible values of selectivity. An AUC of close to 1 means the test is quite effective in distinguishing between classes, while an AUC of close to 0.5 means that the test is little better than random guessing. As mentioned before, different thresholds will give rise to different.

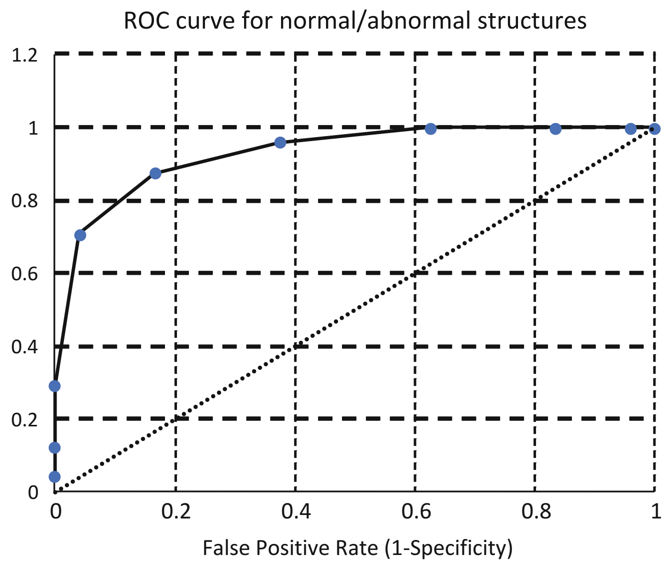


Fig. 4 ROC curve for data shown in Fig. 2

5 A Practical Example: Assessment of NN-GORV-II Algorithm for Structure Identification

The GOR (Garnier–Osguthorpe–Robson) method is widely used to predict the secondary structure of proteins based on information theory and Bayesian statistics [4]. The method has been improved over the years by including larger databases, more detailed statistics, and evolutionary information. The most recent version is known as GOR version 5, or GORV [3, 5].

Figure 5 shows NN-GORV-II test scores (referred to here as "cut scores") of a sample of 10,772 secondary structures (7626 coil and 3146 non-coil). The same data is shown numerically in Table 1 in the first three columns. (Note that as discussed above, strand and helical were grouped together into a single class.)

In the calculation of FPV and TPV, we will consider non-coil as a "positive" result. So, for example, if we take the value 3 as cut point, then all values above 3 are taken as positives (i.e. non-coil). From the table we may calculate that $7626 - 544 - 625 = 6457$ coil structures are falsely identified as positives, so the false positive rate (FPR) for cut point 3 is $6457/7626 = 0.847$. On the other hand, $3146 - 33 - 45 = 3068$ out of 3146 non-coil are correctly identified as positives, giving a TPR of $3068/3146 = 0.975$. The fourth and fifth columns of Table 2 are computed in this manner from the numbers in columns 2 and 3. The area calculation in Table 2 is computed using the trapezoid rule. For example, the area from cut point 5 to cut point 6 is equal to the difference in FPR values times the average TPR value, or

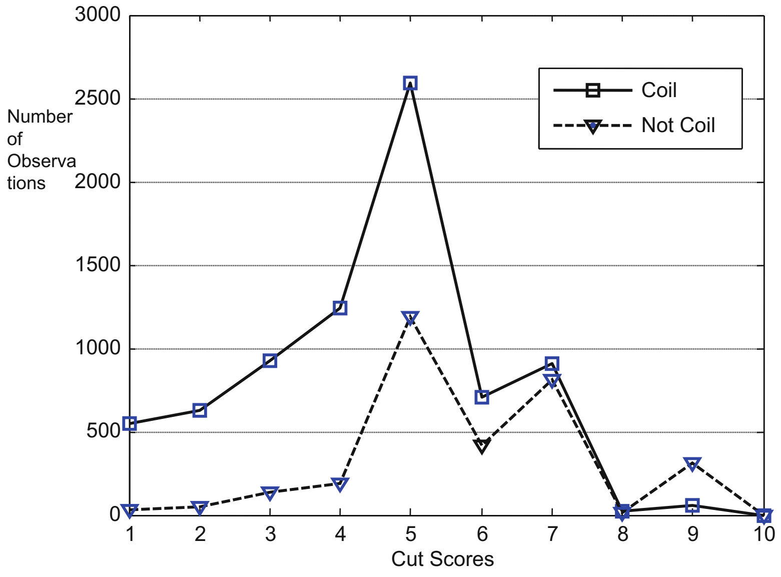


Fig. 5 The cut scores of coil and non-coils secondary structure states predicted by the NN-GORV-II algorithm using Method V reduction scheme

Table 2 Data from Fig. 5 in tabular form, together with calculated false positive rate, true positive rate, and AUC areas

Test value (cut point)	Number of coil	Number of non-coil	FPR	TPR	Area
1	544	33	1	1	0
2	625	45	0.929	0.990	0.071
3	929	139	0.847	0.975	0.081
4	1244	185	0.725	0.931	0.116
5	2588	1187	0.562	0.872	0.147
6	710	415	0.222	0.495	0.232
7	912	814	0.129	0.363	0.040
8	18	14	0.010	0.104	0.028
9	56	314	0.007	0.100	0.000
10	0	0	0.000	0.000	0.000
Totals	7626	3146			0.715

$(0.562 - 0.222)\cdot(0.872 + 0.495)/2$, which yields the value 0.232 in row 6 of the last column. The summation of the nine scores areas represents the total area under the curve (AUC), which is a measure of the prediction accuracy. The AUC of this test as shown in the table is 0.7151.

A rather complicated expression for the standard error of the AUC estimate may be found in [7]. "pROC" is a software package which can be loaded within the R statistics programming language, which can calculate confidence intervals for the AUC. For the data in Table 2, the standard error works out to 0.0057.

Figure 6 shows the ROC curve obtained by plotting the points in columns 4 and 5 of Table 2. Compared with the ROC curve in Fig. 4, we may see clearly that this method has less predictive power than the normal/abnormal structures identification. This should not be surprising because the overlap between the areas under the normal and abnormal cut-point curves in Fig. 5 is much greater than that between coil and non-coil cut-point curves in Fig. 2.

The AUC value of 0.72 may be interpreted as the average TPR over all possible FPR's. Note that the TPR for FPR = 1/2 is equal to 0.8. In fact, it will always be true for any ROC curve that the TPR for FPR = 1/2 will be greater than the AUC. The value 0.72 is consistent with what has been reported by. This result is consistent with the correlation coefficients of the NN-GORV-II method described in our previous work [6].

6 Summary

We have explained the calculation and interpretation of ROC curves as used in binary classification and have demonstrated their use in the classification of protein secondary structures.

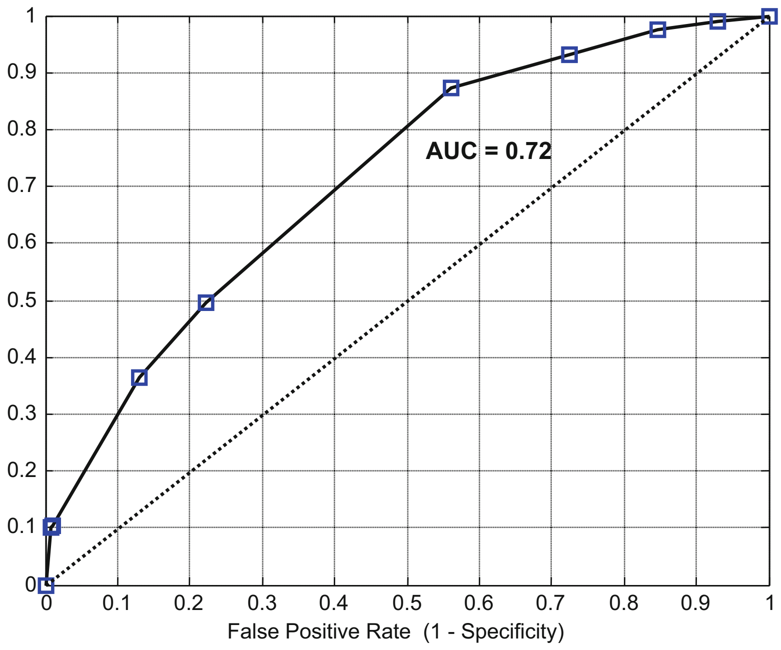


Fig. 6 ROC curve for the data shown in Fig. 5 and tabulated in Table 2

References

1. A.H. Fielding, J.F. Bell, A review of methods for the assessment of prediction errors in conservation presence/absence models. Environ. Conserv. **24**(1), 38–49 (1997)
2. D.J. Hand, W.E. Henley, Statistical classification methods in consumer credit scoring: A review. J. R. Stat. Soc. A. Stat. Soc. **160**(3), 523–541 (1997)
3. A. Kloczkowski et al., Combining the GOR V algorithm with evolutionary information for protein secondary structure prediction from amino acid sequence. Proteins **49**, 154–166 (2002)
4. J. Garnier et al., Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins. J. Mol. Biol. **120**, 97–120 (1978)
5. T.Z. Sen, R.L. Jernigan, J. Garnier, A. Kloczkowski, GOR V server for protein secondary structure prediction. Bioinformatics **21**(11), 2787–2788 (2005)
6. S.O. Subair, S. Deris, Predicting protein secondary structure using artificial neural networks and information theory, in *Application of Agents and Intelligent Information Technologies*, ed. by V. Sugumaran, (Idea Group, USA, 2007), pp. 337–362
7. C. Cortes, M. Mohri, Confidence intervals for the area under the ROC curve. Adv. Neural Inf. Proces. Syst. **17**, 305–312 (2005)

Budget Reconciliation Through Dynamic Programming

Tad Laver, Lucas Brandt, and Christopher Thron

1 Introduction

1.1 Discrepancies in Military Accounting

"*U.S. Army fudged its accounts by trillions of dollars, auditor finds*" reads the banner headline of a 2016 article in Reuters' U.S. web page [1]. The situation described by the article was not necessarily mismanagement of funds, but rather poor bookkeeping due to incomplete records pertaining to expenditures. Here we present a more detailed explanation of the actual situation.

U.S. Army brigade comptrollers who oversee accounts have access to their brigades' daily "commits" and "obligations." "Commits" are like purchase orders: they represent expenses that have been authorized and the order sent to the supplier, but the money has not yet been spent. "Obligations" are actual withdrawals from the brigade's account. Every valid obligation has necessarily been previously committed, but the commit may have occurred several days previously.

Of course, commits and obligations are both important information, and are essential to balancing the budget. Unfortunately, the Army lacks a comprehensive system to correlate specific commits to their corresponding obligations, since only the total daily amounts are recorded. This is complicated by the fact that the total commits do not exactly match the total obligations, because other budgetary items such as credits for returned machinery and gasoline expenditures also affect the record of commits. The disconnect between commits and obligations produces a perpetual uncertainty in the current status of the budget, because it is unknown how much of the remaining funds in the account have already been spoken for. This

T. Laver (✉) · L. Brandt · C. Thron
Texas A&M University-Central Texas, Killeen, TX, USA

S. Subair, C. Thron (eds.), *Implementations and Applications of Machine Learning*, Studies in Computational Intelligence 782,
https://doi.org/10.1007/978-3-030-37830-1_12

uncertainty can lead to budget imbalances, and at the end of the fiscal year this may lead to overspending or unspent funds which are then frozen until they can be accounted for.

Given that a comprehensive system for budget reconciliation is lacking, we may develop an algorithm that infers the correspondence between commits and obligations as accurately as possible. This is a very complex problem because of the large number of expenditures involved. It may be characterized as an optimization problem, since we want to find the "best" or most likely match between commits and obligations.

1.2 Dynamic Programming Overview, and a Simple Example from Biochemistry

Dynamic programming is a powerful technique for solving very large optimization problems. This was first developed by R. Bellman at the RAND Corporation in the 1950s [2]; since then it has been applied to a wide variety of applied fields in engineering and computer science [3–6]. Dynamic programming can be characterized as a "divide and conquer" approach: an optimal solution is found by successively restricting the set of possible solutions, so that the search is conducted very efficiently.

Before explaining the dynamic programming algorithm used in budget reconciliation, we first present a simple example that elucidates the principles and procedures underlying dynamic programming.

The Viterbi algorithm is a dynamic programming algorithm which determines a most probable sequence of states or events based on partial information. For each successive state in the sequence, it finds a limited set of possible best sequences up to that state. After the last state has been processed, the best of the remaining sequences is chosen, which gives an overall best sequence based on all the information provided. The Viterbi algorithm is widely applied in different fields, most importantly in digital communications where it played a crucial role in the development of digital cellular telephony [7]. The following example is an application of the Viterbi algorithm to molecular biology [8].

Deoxyribonucleic acid (DNA) molecules in a living cell contain the genetic information that is used to duplicate the cell during cell reproduction. DNA molecules consist of two linked strands, and each strand consists of a sequence of nucleotides. There are four nucleotides (Adenine, Cytosine, Guanine, and Thymine, represented as A, C, G, T) and they can appear in virtually any order: for example, ACTCCTCCGTAAGTG . . . represents a possible sequence.

Different sections of a DNA molecule can be roughly identified as being in one of two states.

The two states in this case are referred to as H and L, or high and low GC ratio, respectively. GC ratio deals with guanine-cytosine content in a sequence and is a measure of what ratio of the DNA sequence is made of guanine and cytosine. A DNA sequence can move between a high and low GC ratio many times within its chain, but it will also have an overall GC content. In general, longer chains will tend to have a high GC content and shorter chains will have a low GC content.

Figure 1 shows the amino acid frequencies and transition probabilities between high and low states in a typical DNA sequence. The two boxes labeled H and L give the frequencies of the different nucleotides corresponding to the two states: the H state has a higher probability of producing G and C nucleotides, and the L state has a higher probability of producing A and T nucleotides. The arrows emerging from the box representing the H state indicate that if the current nucleotide is in a H segment, then there is a probability of 0.5 that the next nucleotide is also in the H state, and a probability of 0.5 that the state switches to L. On the other hand, if the current nucleotide is in a L segment, according to the arrows emerging from the "L" box there is a probability of 0.6 that the next nucleotide is also in the L state, and a probability of 0.4 that the state switches to H. The arrows pointing from the "Start" box indicate that is an even chance that the chain starts in H or L states, as indicated in the top box in the figure.

We do not have any way of directly observing which state a DNA sequence is in at any moment: for this reason, H and L are called *hidden states*, and this type of model is called a *Hidden Markov Model (HMM)*. Although it is not possible to precisely determine the sequence of H's and L's for a given DNA chain, using the probabilities given in Fig. 1 it is possible to compute the HL sequence that is most probable, given the observed nucleotide sequence. The Viterbi algorithm enables us to calculate this most probable sequence in a highly efficient way, as we shall see shortly.

In order to clarify the calculations involved, we will compute the probability that the nucleotide sequence TAC is observed and at the same time the hidden HL sequence is LLH. The initial probability of starting in L and observing nucleotide T is (.5)(.3) = .15. The probability of transitioning from L to L and observing nucleotide A is (.6)(.3) = .18. The probability of transitioning from L to H and observing nucleotide C is (.4)(.3) = .12. Since these three successive transitions are independent of each other, we may multiply them together to get the net

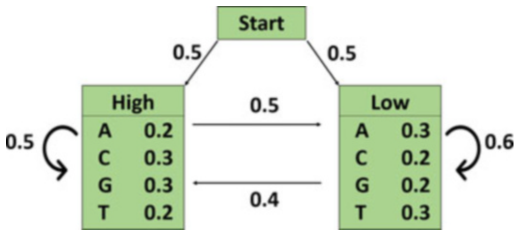


Fig. 1 Schematic representation of High and Low states, showing transition probabilities between states and probabilities of getting each nucleotide while in each state (after Borodovsky and Ekisheva [8])

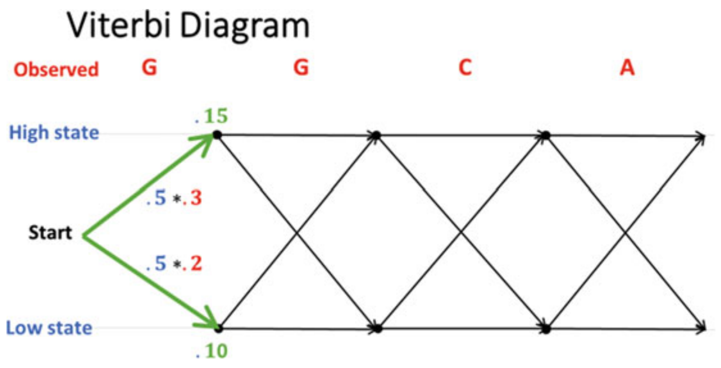


Fig. 2 Calculations for initial state probabilities. State H and observed nucleotide G has a higher probability than state L and G, but at this point we are not ready to make a final decision

probability. Therefore, the net probability of getting TAC along with LLH is (.15)(.18)(.12) = .00324.

With the calculations clarified, let us now find the HL sequence that has the maximum probability, given the observed nucleotide sequence. Let A and B represent the observed nucleotide sequence and the hidden HL sequence, respectively. Then $P(B|A)$ represents the conditional probability that we would like to maximize. This conditional probability is related to the probability that A and B are both true as follows: $P(B|A) = P(A \text{ and } B)/P(A)$. Since A is known, it follows that $P(A)$ is a fixed number. Hence maximizing $P(B|A)$ is equivalent to maximizing $P(A \text{ and } B)$ because the two quantities differ by a constant factor. Hence in the following we will address the problem of maximizing $P(A \text{ and } B)$, i.e., finding the HL sequence out of all possible HL sequences that gives maximum probability when paired with the observed nucleotide sequence.

We may represent the Viterbi approach to this calculation using a *trellis diagram*, as shown in Fig. 2. The observed nucleotide sequence GGCA... is used to label the stage in the trellis. The upward-sloping green arrow indicates the case where the first nucleotide is G, and the first HL state is H: the probability of this case is $P(\text{start} \rightarrow \text{H}) \cdot P(\text{G}|\text{H}) = (0.5)(0.3) = 0.15$, based on the numbers given in Fig. 1. The downward-sloping green arrow shows the case where the first nucleotide is G and the first HL state is L: the probability of this case is found to be 0.1.

The stages in the trellis are computed successively. Figure 3 shows the calculations for the second stage. This time, four transitions are computed: H → H, H → L, L → H, and L → L. The transition probability for H → H is multiplied by the probability that the first HL state is H, giving the result that $P(\text{HH}) = (0.15)(0.15) = 0.225$. The probabilities $P(\text{LH})$, $P(\text{HL})$, and $P(\text{LL})$ are computed similarly, yielding 0.012, 0.015, and 0.012, respectively. In this case, $P(\text{HH}) > P(\text{LH})$, so of the possible 2-letter sequences ending in H, HH is preferred. It follows that LH may be eliminated, since all sequences beginning LH... will have lower probabilities than the corresponding sequences beginning HH.... Similarly,

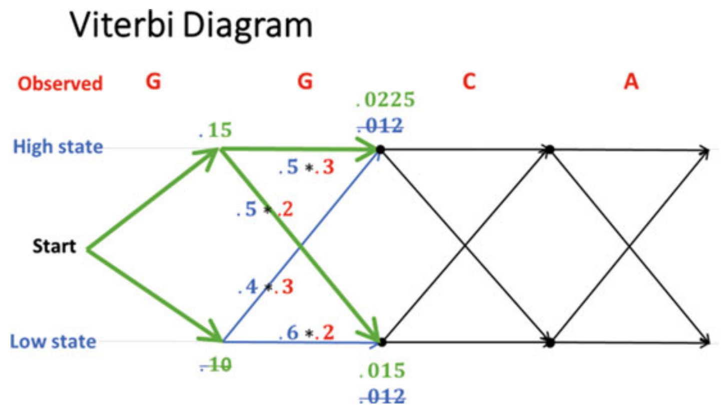


Fig. 3 Second set of calculations, and state L and observed nucleotide A has the highest probability of the four calculated combinations at this stage

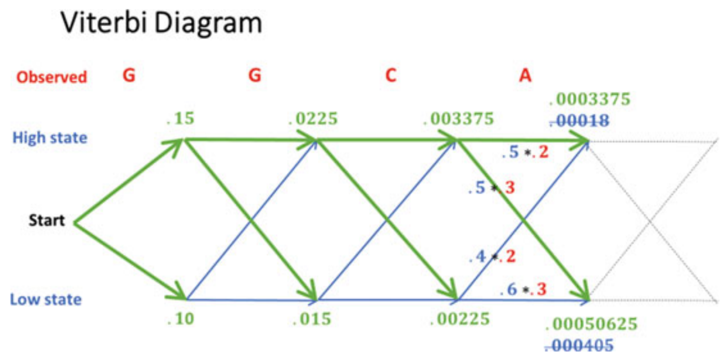


Fig. 4 Calculations for the fourth trellis stage. The surviving HL sequences may be found by following the green arrows backwards from the final L state, which is the final state with the higher probability

of the 2-letter sequences ending in L, $P(\text{HL}) > P(\text{LL})$, and HL is preferred, so LL may be eliminated as a possibility. As a result, in Fig. 4 the probabilities corresponding to LH and LL have been crossed out. The green arrows at the second stage indicate the transitions that correspond to the two surviving HL sequences, HH and HL.

Figure 4 show the results from the fourth and final stage in the computation of this simple 4-letter example. The probability for the path ending in L is greater than for the path ending in H, so the lower ending is chosen. Tracing back along the green arrows, we find that HHHL is the HL path which, when combined with the observed nucleotide sequence GGCA, has the highest probability.

We may compute the number of calculations that were required to find this optimal HL path. Moving from start to the first state required computing two transition probabilities. Subsequently, each stage required 4 transition probabilities.

This makes a total of $2 + 4 + 4 + 4 = 14$ calculated probabilities. This is a savings of 2 over calculating the probability for all 16 HL sequences of length four, which seems insignificant. However, the savings mount up very quickly when the number of stages increases. Using the Viterbi algorithm, each additional stage requires 4 additional calculations: it follows that n stages requires $4n - 2$ probability calculations. On the other hand, the number of HL sequences of length n is 2^n, which grows large quickly. For example, if we have a DNA sequence of length 100, the Viterbi algorithm requires less than 400 calculated probabilities, while an exhaustive calculation of all HL sequence probabilities would require calculation of 2^{100} probabilities!

1.2.1 Budget Reconciliation with Dynamic Programming

Our goal was to derive, implement, and verify a dynamic programming algorithm that can reconcile commits and obligations in Army brigade budgets. The requirements for the algorithm were determined to be the following. First, it must first be able to identify the most likely match between commits and obligations for budget reconciliation purposes. Second, the algorithm must also be able to estimate the probability distribution of commit-to-obligation delays for predictive purposes. Finally, it must also be easily adaptable to a wide range of scenarios.

For our original implementation, we made certain assumptions based on one author's (LJB) personal experience with managing brigade budgets. As indicated above, we assumed that the commits and obligations for a given unit are reported as separate streams. We also assumed that each commit obligates as a whole within a given time window of w days (which was taken as one working week of 6 days). Although this is an approximation, it is fairly accurate because commits are usually dominated by a single large expenditure (e.g., an aircraft engine) which will naturally be paid for all at once. (It is possible to relax this assumption and perform further optimization using linear programming, but this is beyond the scope of this chapter.)

Finally, we assumed that return credits and untracked expenses such as fuel introduce Gaussian noise into the system, so that there is a normally distributed (i.e., Gaussian) discrepancy between commits obligated on any given date and the recorded obligation for that date. The mathematical implications of this assumption are as follows. Let us set:

$X_n :=$ (sum of commits that obligate at time n) $-$ (recorded total obligation at time n).

The assumption of Gaussian noise implies that the probability density p of X_n is given by a Gaussian distribution:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}},$$

where σ is the (unknown) standard deviation of the discrepancy.

Since all of the X_n's are assumed to be independent, it follows that probabilities multiply and the probability that $X_1, X_2, \ldots, X_N$ take values given by the sequence $(x_1, \ldots, x_N)$ is.

$$p(x_1, x_2, \ldots, x_N) = p(x_1) \ldots p(x_n) = \left(\frac{1}{\sqrt{(2\pi\sigma^2)}} \right)^n e^{-\frac{x_1^2 + \cdots + x_N^2}{2\sigma^2}}.$$

We want to find the most likely sequence $x_1, \ldots, x_N$. This implies that we want to maximize the function $p(x_1, x_2, \ldots, x_N)$, which is equivalent to maximizing its logarithm since log is an increasing function. We have:

$$\log p(x_1, x_2, \ldots, x_N) = a - b\left(x_1^2 + x_2^2 + \cdots + x_N^2\right),$$

where a and b are constants. It follows that maximizing $p(x)$ is equivalent to minimizing the sum of squares $x_1^2 + x_2^2 + \cdots + x_N^2$. In summary, in order to find the most likely matching between commitments and obligations, we should find the possible matching that minimizes the sum of squared differences between observed and inferred obligations for the entire process. For a year-long budget this problem would have hundreds of variables, so the exhaustive approach of trying all possible matchings is wildly impractical. On the other hand, dynamic programming is eminently suited for finding solutions, as we shall see.

2 Methods

2.1 Dynamic Programming Algorithm Step-by-Step Description

In order to create a trellis as we did with the nucleotide sequence example, we will need to define a set of states of the system that undergo transitions from day to day. As in the nucleotide example, each state should encapsulate all relevant (but hidden) information about commits and obligations that influence transitions to the next state, and which can be used to compute transition probabilities. Notice that on day n, the only possible obligations consist of the commits that were made w or fewer days previously: in other words, only the commits made between $n - w$ and $n - 1$ which have not yet been obligated may be obligated at time n. It follows that the only relevant information needed for determining the possible commits on day n is the set of unobligated commits for the time interval $[n - w, n - 1]$. This information may be encoded as a string of w bits, where each obligated (resp. unobligated) commit in the interval is represented by a 0 (resp. 1). So, for example, if $n = 10$ and $w = 4$, then the string 0110 indicates that the commits at times 6, 9 have been obligated and the commits at times 7, 8 remain unobligated.

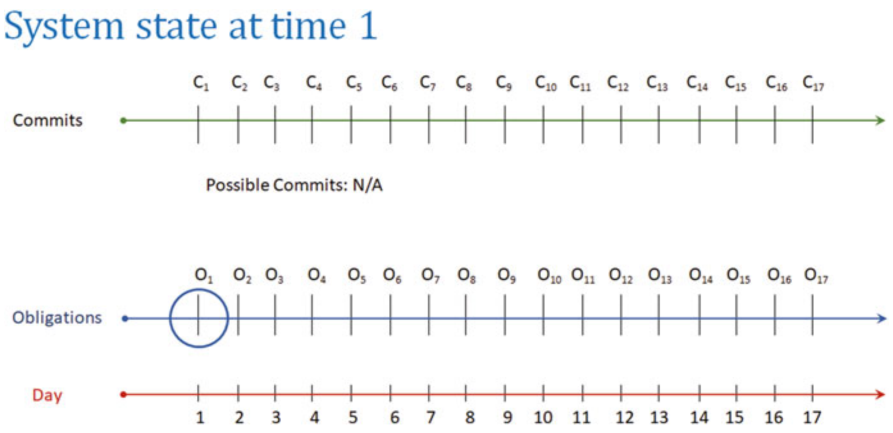

System state at time 1
Commits
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17
Possible Commits: N/A
Obligations
O1 O2 O3 O4 O5 O6 O7 O8 O9 O10 O11 O12 O13 O14 O15 O16 O17
Day
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

Fig. 5 On day one, there is no possibility of a commit yet, and the "window" is nonexistent

Here we give a step-by-step example to show the transition between states over time. Subsequently we will construct the trellis, which is somewhat more complicated than the trellis in Fig. 1. In this example we will use $n = 6$.

The process begins at $n = 1$ as shown in Fig. 5. The state of the system is the empty set, because there are no prior commits that can be obligated or unobligated.

At time $n = 2$, there is only one prior commit, namely the commit at time 1. This commit cannot have been previously obligated, since a commit cannot be obligated the same day. Therefore, the only possible state at $n = 2$ is the single bit 1.

At time $n = 3$, the prior commits occurred at times 1, 2. The commit at time 2 must be unobligated, but the commit at time 1 could be either obligated or unobligated. It follows that the possible states at $n = 2$ are 01 and 11.

If we skip down to $n = 6$, we find that there are 5 prior commits, as shown in Fig. 6. The commit on day 5 must be unobligated, but the other four may or may not be obligated. It follows that there are 16 possible states at $n = 6$, as listed in the figure.

The state size grows larger from $n = 1$ to $n = 7$. When $n = 7$, the state consists of 6 bits, where the final bit is 1: this implies that there are $2^5 = 32$ possible states. Since we are using $w = 6$, it follows that 6 bits is the maximum state size and for $n \geq 7$, the window remains the same size and shifts as n increases (see Fig. 7).

We are now ready to construct the trellis expressing the transitions from state to state, in analogy to Fig. 2. Given each state at time n, there is a limited set of possible previous states at time $n - 1$. To illustrate this, let us look at a simplified case with only four bits ($w = 4$), and we suppose the state at time n is 0101 (the transitions will be the same for any n, as long as $n > 4$). In this case, the bit string 0101 indicates that times $n - 4$ and $n - 2$ are characterized by "0" bits, meaning that the commits at these times have been obligated before time n. On the other hand, times $n - 3$ and $n - 1$ have "1" bits, indicating that the commits at these times have not yet been obligated before time n. This implies that the commit at time $n - 3$ was also unobligated at time $n - 1$. Therefore, the second to rightmost bit in the state at $n - 1$

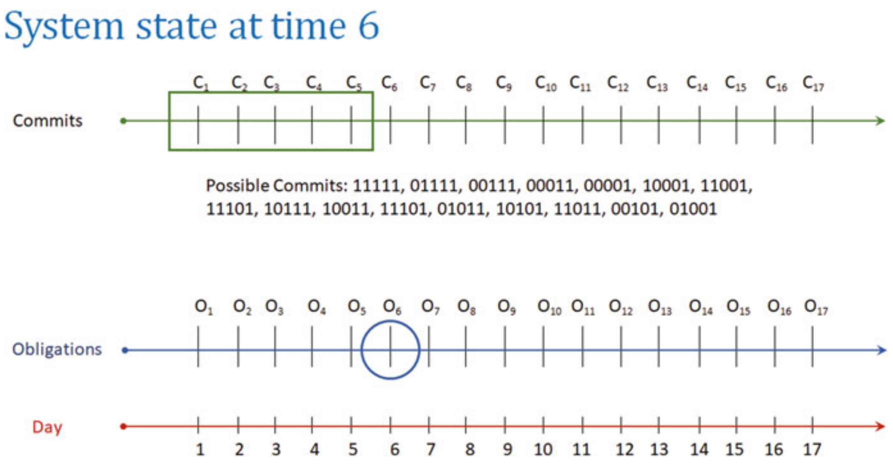

System state at time 6
Commits
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17
Possible Commits: 11111, 01111, 00111, 00011, 00001, 10001, 11001, 11101, 10111, 10011, 11101, 01011, 10101, 11011, 00101, 01001
Obligations
O1 O2 O3 O4 O5 O6 O7 O8 O9 O10 O11 O12 O13 O14 O15 O16 O17
Day
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

Fig. 6 On day 6 the window is 5 days long, with 16 possible commits

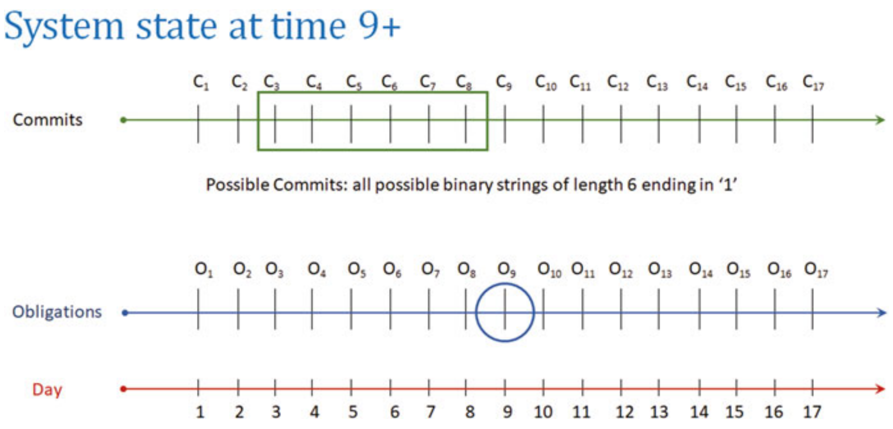

System state at time 9+
Commits
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17
Possible Commits: all possible binary strings of length 6 ending in '1'
Obligations
O1 O2 O3 O4 O5 O6 O7 O8 O9 O10 O11 O12 O13 O14 O15 O16 O17
Day
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

Fig. 7 On every day from 7 on there will be 32 possible combinations with each possible commit being a bit string 6 long with the last entry being a 1

must be a "1." We also know that the rightmost bit in any state is a "1." As to the two leftmost bits of the state at time n-1, they may or may not have been obligated. Putting all this information together, we may conclude that the state at time $n - 1$ must have had the form **11, where '*' indicates the bit could have been either 0 or 1 depending on whether or not the corresponding commit obligated before n. In Fig. 8 we have drawn segments joining possible source states for destination state 0101.

Likewise, for each destination state we may draw segments that represent possible transitions. The result is the trellis shown in Fig. 9. In the trellis, only half of the possible segments from source to destination are drawn—hence use of the trellis can reduce calculation by ½.

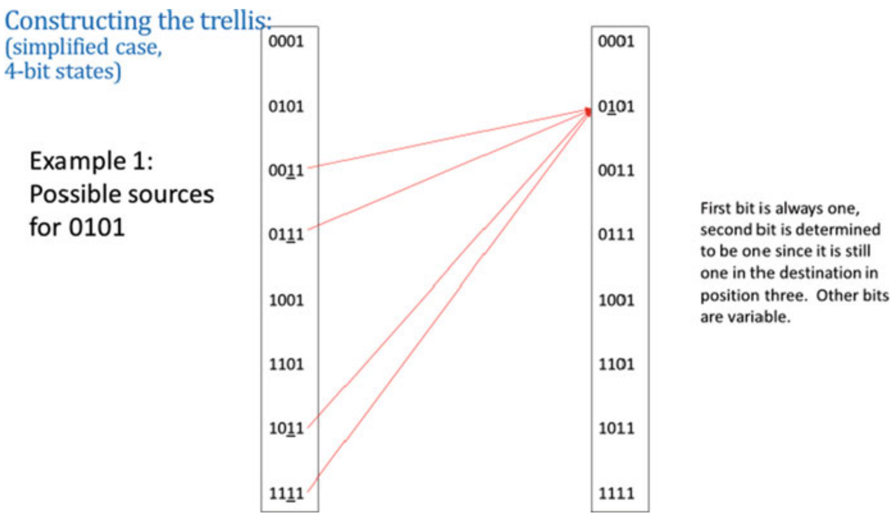

Constructing the trellis:
(simplified case,
4-bit states)
Example 1:
Possible sources
for 0101
0001
0101
0011
0111
1001
1101
1011
1111
First bit is always one, second bit is determined to be one since it is still one in the destination in position three. Other bits are variable.

Fig. 8 Since we know that the destination is 0101, we limit the possible sources from 8 to 4 in this case

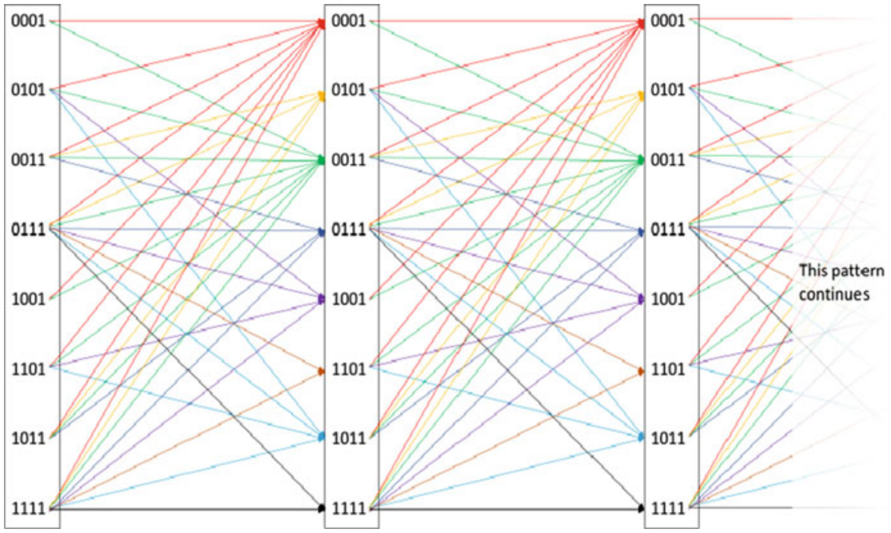


Fig. 9 This trellis shows the possible sources for each element in each state. Some elements are very restrictive while others are not restrictive at all

Each transition segment in the trellis that joins a state at time $n-1$ to a state at time n corresponds to a particular set of obligations at time n. For example, Fig. 10 includes the transition $0111 \rightarrow 0011$, which indicates that the commits at times $n-4$ and $n-3$ were obligated at time $n-1$. We may use this commit information to determine the value of the deviation x_n, and thus determine the corresponding

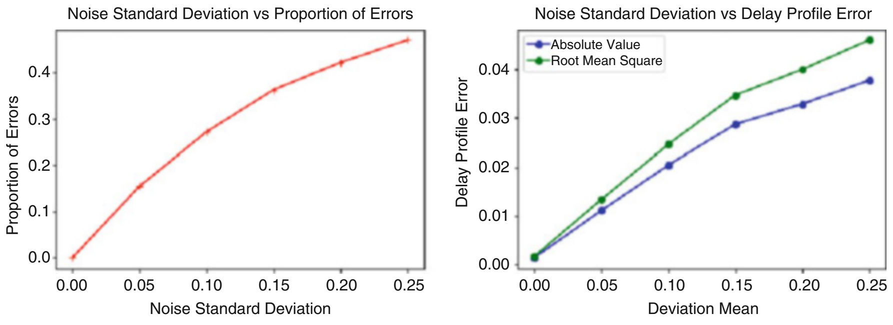


Fig. 10 Proportion of erroneous commit-to-obligation assignments (*left*) and mean absolute error and root mean squared error in delay profile (*right*) as a function of noise standard deviation

probability $p(x_n)$, which may in turn be used to calculate the probability of the path through the trellis that ends with that particular transition.

We must make special mention of the case where the leftmost bit of the state at time $n-1$ is a "1," indicating that the commit at time $n-w$ has not yet been obligated before time n. Whether or not this commit obligates at time n, the system will transition to the same state at time n, because (as we have mentioned previously) the leftmost bit of the state drops off at the next time. So, in fact, each segment in the trellis whose left endpoint is a state beginning with "1" represents two different possible transitions. The algorithm deals with this ambiguity by evaluating $p(x_n)$ in both cases, and choosing the larger alternative. If it turns out that the leftmost commit does not obligate, it can never be obligated in the future, because it has passed out of the obligation window. When this happens, the algorithm has identified a potential unobligated commit, which is a budget discrepancy that should be further investigated.

2.2 Code Structure

We programmed the algorithm in the open-source software Python, because of its high-level mathematical capabilities as well as its good execution speed. At top level, the code structure is as follows:

1. Initialization of parameters and data structures.
2. Generation of simulated commits and obligations.
3. Loop: dynamic programming process.
4. Recovery of solution and output of statistics.

These four stages in the code are described in more detail below.

2.2.1 Initialization

The following initializations were made:

- Parameters for simulated commits: Total number of days (T), probability of a commit on any given day (`prob_commit`), commit mean size.
- Parameters for simulated obligations: Window size (w); probability distribution of commit-to-obligation delays (i.e., delay profile); mean and standard deviation of unmatched obligations (credits and incidental expenses).
- Data structures for dynamic programming:
    - List of states. This is a list of length 2^{w-1}, where each list entry is an array of w binary bits. All possible arrays of w bits are included which have final bit 1. The arrays are generated in binary numerical order. This list provides a "dictionary" that enables translation from integer to binary representations of states.
    - Histories of previously obligated commits corresponding to each current state. These histories represent the pattern of obligated commits that minimize the sum of squared differences between daily obligated commits and actual observed obligations. Each current state has its own unique history, just as in the nucleotide example. The histories comprise a list of 2^{w-1} items corresponding to the 2^{w-1} possible states. These 2^{w-1} items are themselves lists with n entries, where n represents the current time: the kth entry is an integer between 0 $2^w - 1$ and (inclusive) which represents the commits that obligate at time k. For example, suppose $w = 5$ and the 29th entry of the ninth in-list is 25. This implies that in the history of state 9 (representing the binary string 01001) the commits that are obligated at time 29 are indicated by the binary representation of the integer 25 (11001). Initially, the 2^{w-1} histories are all 1-entry lists with the single entry 0, since at time $n = 1$ there are no prior commits and hence no obligations. The histories are updated at each trellis stage: at stage n, a new entry is added to each history, representing commits that are obligated at time n.
    - List of current values associated with states. The current value of a state is given by the sum of squared differences between daily obligated commits determined by the state's history and observed daily obligations. The list of current values is an array of real numbers of length 2^{w-1}, initialized to all zeros. This list is updated at each time step, when the values of states are updated at each trellis stage.

2.2.2 Generation of Simulated Commits and Obligations

Commits were generated independently for each day. For any given day, a commit was generated with probability `prob_commit`. Each commit was generated as an exponential random variable with the given mean (the standard deviation of an exponential random variable is equal to the mean).

Obligations were generated as follows. For each commit, a delay is generated according to the specified delay profile. The total obligation on any given day is the sum of delayed commits that fall on that day, plus a random noise that represents untracked credits and incidental expenses.

2.2.3 Loop over N: Dynamic Programming Process

The steps in each iteration in the dynamic programming process are as follows:

- Initialize the destination state values as an array of size 2^{w-1}, where each value in the array is initialized to a very large negative number (these values will be replaced by new values based on the trellis calculations.
- Loop through source states:
    - Loop through all transitions from possible source states.
        - Compute updated value (source state value + transition value).
        - If updated value is less than the transition destination state's current value.
            - Replace destination state value with computed value.
            - Append to source state history an additional integer that represents the commits obligated at time *n* as indicated by the transition.
            - Replace destination state history with this updated source state history.
        - End (If).
    - End (loop through transitions).
- End (loop through source states).
- Replace all source state values with destination state values.
- Replace all source state histories with destination state histories.

2.2.4 Recovery of Solution and Output of Statistics

After the algorithm has looped through all time steps, the state with the minimum value is chosen, and the history of that state is read to determine the obligation times of all commits. These obligation times are compared with the actual obligation times, and these results are used to compute error rates.

3 Results

The main results were obtained via simulation. Some results were also obtained by applying the algorithm to actual data, as described below.

Table 1 Baseline parameters used in simulations

Parameter type	Parameter	Value
General	Simulation time (days)	100,000
Commit Specifications	Prob. of commit on any given day	1
	Daily commit mean	1
	Daily commit standard deviation	1
Obligation specifications	Window size	6
	Delay profile (probability distribution of delays)	$\left[\frac{1}{12}, \frac{1}{6}, \frac{1}{4}, \frac{1}{4}, \frac{1}{6}, \frac{1}{12}\right]$
	Noise mean (expected sum of untracked, obligated daily expenses)	0
	Noise standard deviation	0.05

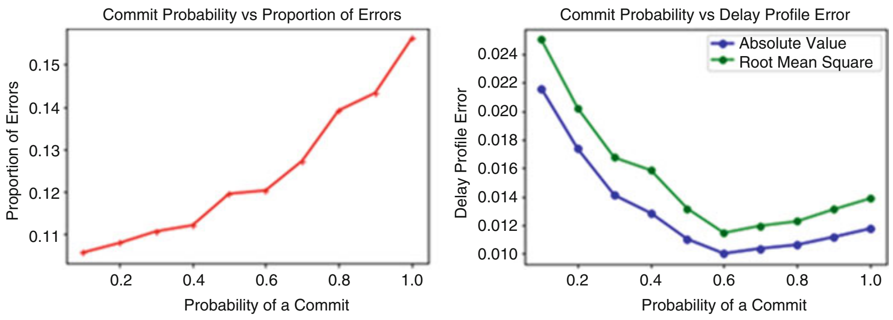


Fig. 11 Commit-to-obligation assignment error probability (*left*) and mean/root mean squared error of delay profile estimate (*right*) as a function of the daily probability of a commit

3.1 Simulation

Simulations were run to verify the model's accuracy. Baseline parameters are listed in Table 1. For all simulations, parameters are set to baseline values unless otherwise specified.

Figure 10 shows the dynamic programming algorithm's ability to match commits to obligations and to estimate delay profiles as a function of noise standard deviation (recall that in practice, noise is due to untracked credits and various untracked expenses which are a small fraction of total expenditures). When the noise level is zero, errors are also zero, indicating that the algorithm performs perfectly on a clean record where all daily commits and obligations are recorded. Not surprisingly, both assignment errors and delay profile estimation error increase as the noise standard deviation increases. When the noise standard deviation reaches 0.25 (i.e., roughly ¼ of expenses are untracked) up to 50% of obligations are incorrectly identified, but the mean absolute delay profile error is still rather small (about 4%).

Figure 11 shows error rates and delay profile estimation errors for different commitment frequencies. As described earlier, the model presumes either a single commit or no commit on each day, so the commitment frequency may vary from 0 (no commits at all) to 1 (a commit occurs every day). On days when commits occur, the commit size is chosen randomly according to an exponential distribution with parameters as in Table 1. The figure shows that the proportion of errors increases as the frequency of commits increases: this may be attributed to the fact that as commits become more frequent, the number of possible matchings between commits and obligations increases and hence the greater possibility for error. On the other hand, the delay profile error initially decreases as the commit probability increases, and reaches a minimum at around $p(\text{commit}) = 0.6$. The second figure shows that the mean absolute delay profile error is consistently about 90% of the root mean squared error: in comparison, for a Gaussian distribution the mean absolute error is about 80% of the root mean squared error.

3.2 Application of Algorithm to Real Budget Data

In addition to establishing the algorithm's accuracy on simulated data, we also verified that the algorithm is able to identify budget errors on real data. One author (L.B.) was a brigade comptroller, and applied the algorithm to data from his own brigade. The algorithm uncovered a discrepancy between commits and obligations in the data. Upon further investigation, \$2,480,000 in commits were found to be unaccounted for. Fortunately, he was able to act upon this information within the same fiscal year and rectify the brigade's account.

4 Conclusions

The dynamic programming method developed in this paper has been shown via simulation and practice to reliably give largely correct matchings between commits and obligations, as well as fairly accurate delay profiles. The method can be extended directly to more complicated situations, for instance, when the delay profiles are time-dependent or when there are multiple types of commits with different delay profiles. The code's modular structure facilitates modifications to accommodate more complicated situations. The assumption that any given day's commits obligate as a whole may be relaxed by a two-stage algorithm. In the first stage, the best matching with undivided commits is found (as in the current algorithm). In the second stage, inferred obligations that differ significantly from observed obligations may be optimally divided among neighboring days by solving a linear programming problem.

The use of binary strings to represent states is a technique that can also be applied to other types of problems, for instance, scheduling of jobs when the jobs must be completed within a limited time window.

References

1. S.J. Paltrow, U.S. Army fudged its accounts by trillions of dollars, auditor finds. [online] Reuters (2016). Available at https://www.reuters.com/article/us-usa-audit-army/u-s-army-fudged-its-accounts-by-trillions-of-dollars-auditor-finds-idUSKCN10U1IG [Accessed 18 Dec. 2018]
2. R. Bellman, *Dynamic Programming* (Princeton University Press, Princeton, 1957)
3. D.P. Bertsekas, *Dynamic Programming and Optimal Control* (Athena Scientific, Belmont, 2005)
4. C. Freak, Top 50 dynamic programming practice problems. Noteworthy—J. Blog. (2018). [online] Available at: https://blog.usejournal.com/top-50-dynamic-programming-practice-problems-4208fed71aa3 [Accessed 24 October 2019]
5. International Federation of Operations Research Societies (IFORS). IFORS tutorial: dynamic programming (n.d.). [online] Available at: http://ifors.org/tutorial/category/dynamic-programming/ [Accessed 7 Oct. 2019]
6. J. Kleinberg, E. Tardos, Algorithm design. Pearson Education India (2006)
7. A.J. Viterbi, A personal history of the Viterbi algorithm. IEEE Signal Process. Mag. **23**(4), 120–142 (2006)
8. M. Borodovsky, S. Ekisheva, *Problems and Solutions in Biological Sequence Analysis* (Cambridge University Press, Cambridge, 2006)

Index